AUTOMATED SOFTWARE UPGRADE UTILITY

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit U.S. Provisional Patent Application No. 60/155,502 filed September 22, 1999, and is incorporated by reference herein.

TECHNICAL FIELD

Embodiments of this invention relate to automatically upgrading software.

BACKGROUND AND SUMMARY

Automatic upgrade utilities are known in the prior art. However, prior art automatic upgrade utilities are not known to have provided firmware upgrades across an entire product line for units comprised of a plethora of different characteristics. In addition, prior art on automatic upgrade utilities are not known to operate across any network transports. Prior art on automatic upgrade utilities have rarely operated in the automatic data collection ("ADC") device platform environment.

Many suppliers customize their products according to customer requirements. This is especially true with regard to products related to automatic data collection ("ADC"). Accordingly, the precise characteristics of the products sold by the supplier, even within a single product line, vary greatly. In the prior art, upgrades required a confusing disk swapping process and/or a direct serial connection to the device that required considerable user attention to ensure the upgrade completed successfully. Upgrades frequently consumed several hours, as each detail of the product being upgraded had to be painstakingly checked and disks constantly swapped. Moreover, an operator needed to be present to swap the diskettes. The presence of the operator often meant that during business hours the ADC device platform was unavailable for data collection while an upgrade was being accomplished. Many users were extremely reluctant to perform the installation procedure themselves because they were afraid of

making mistakes and they were also concerned that they did not possess the requisite technical understanding to complete the process successfully.

The following summary lists the issues resolved by implementing the automated upgrade process:

- Reduce the downtime of the device due to an upgrade being performed
- Reduce inputs required from the user to eliminate operator error
- Reduce the time required to perform an upgrade, reduce manpower required
- Increase the odds of having the upgrade process conclude successfully
- Allow for scheduling of the upgrade during off-hours
- Initiate simultaneous upgrades to multiple target devices
- Allow for grouping of devices to reduce redundant operations
- Perform the upgrade over a network to many devices as well as over a serial connection to a single device
- Allow for upgrades to occur from remote, centralized locations
- Support differing scopes of upgrades via the same utility, i.e. install a patch versus install a new version of the operating system
- Provide a single utility that can perform upgrades of a device's operating system, firmware, application and data files.

The automated software upgrade utility allows a customer, product supplier or software vendor to upgrade the operating system, firmware, applications and data files on any product regardless of the product type and characteristics. This upgrade process can be invoked from a remote location or via interaction directly with the target device.

The automated upgrade process is independent of the device hardware platform, operating system, the network transport utilized by the device, and the target device itself.

The automated software upgrade process also allows for modification of the hardware configuration of the target device, *e.g.*, change and reformat a hard drive partition.

The automated upgrade utility may reside at remote sites, such as the user's place of business, and periodically query the product supplier for firmware upgrades.

Alternatively, the upgrade utility may reside with the product supplier and periodically locate remote products and perform the upgrade process.

In both configurations, the software upgrade utility retains the ability to identify all pertinent product characteristics, freeing the developer of the firmware upgrade from concerns regarding installing the upgrade across a wide range of product characteristics and nearly endless possible configurations.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram illustrating a suitable environment for aspects of the invention.

DETAILED DESCRIPTION

The automated software upgrade utility enables a product supplier or software vendor to remotely upgrade the firmware on any of its products, such as automatic data collection ("ADC") device platforms, regardless of product type or product characteristics. The upgrade utility may reside at remote sites, such as the product owner's place of business, and periodically query the product provider for firmware upgrades. Alternatively, the upgrade utility may reside with the product provider and periodically locate products at a product owner's facility and perform the upgrade process. In both configurations, the software upgrade utility retains the ability to identify all pertinent product characteristics, freeing the developer of the firmware upgrade from concerns regarding installing the upgrade across a wide range of products, each having nearly endless possible configurations. The invention is applicable for both ADC device platforms, ADC servers, and associated devices such as printers and radios.

Upgrades may be classified from minor to severe. The minor upgrades are typically small software patches that have been determined not to pose a serious problem to other functionality. In contrast, severe upgrades constitute functionality that must be carefully provided to the product so as not disrupt the product or its functionality.

The Product Supplier or Software Vendor publishes a software update, i.e., patch, upgrade or new release. The published releases can either be pushed down to the software upgrade server or they can be pulled from the published location.

Once the software upgrade utility has received the published release, it can be distributed in one of four methods. The first being, the software upgrade utility can act as a publishing location for other software upgrade servers, i.e., push the software release to other software upgrade servers or allow them to pull the latest software releases from the master software upgrade server.

The second method is to schedule jobs within the software upgrade utility to push the software release to the target device(s), data collection devices registered in IDRS, at the requested time.

The third method involves when devices initially register with IDRS for the first time. If IDRS is configured such that a specific configuration must exist on the device, and software is a component of that standard configuration, then the designated files are automatically transmitted to the target device.

The fourth method involves a device that has an outdated version of the software resident. Upon startup of the device a check is done to ensure that the correct version(s) of the applicable software is resident, if not the device initiates the request for the upgrade to be initiated.

Following are the functional requirements for the automated upgrade utility:

- Allow upgrades or replacement of a device's operating system, firmware, application and data files from a single utility.
- Allow for scheduling of upgrades to occur at designated scheduled times or initiated manually at any time.
- Allow for devices to be organized into logical groups to allow for a single event to initiate upgrades for multiple devices.
- Allow for the upgrades to be performed via a remote, centralized location as well as from the location where the devices are physically located.
- The upgrade process must be independent of the transport network that the device is attached to.
- The upgrade process must support both devices that exist on wired networks as well as wireless networks.
- The upgrade process itself must not involve any user involvement once it has been initiated.

- Software to be upgraded can originate from any location, i.e. CD-ROM, telnet to the hard disk of system running the upgrade process, retrieved from a web site or bulletin board, etc.
- If the nature of the upgrade requires the hardware configuration of the target device to change, then this must not require user intervention. It must be fully automated as well. An example is changing the size of the hard disk partitions or changing the operating system. After the upgrade is completed, the device must be returned back to a fully functional state such that it can be used in a production environment. All configuration parameters and applications must be restored to allow this to happen.

Figure 1 illustrates a server 100 having a firmware upgrade utility 101. The server 100 may be operated by the producer of ADC device platform, for example. The firmware upgrade utility 101 utilizes a transportation network 102 to reach a host 103. The firmware upgrade utility 101 may perform its operations using any transportation network 102. The host 103 may comprise a centralized computing facility for a purchaser of the ADC device platforms products, for example. The host 103 in turn communicates with one or more controllers 104 over a transportation network 107 that in turn communicate with one or more ADC device platforms 105 over a transportation network 108. Each controller 104 typically communicates with one or more ADC device platforms 105. The firmware upgrade utility 101 may also perform its operations using any transportation network 107 and any transportation network 108.

The upgrade utility 101 determines whether the upgrade concerns the controllers 104 and/or the ADC device platforms 105. The upgrade utility 101 locates the communications procedure for communicating with the host 103, the controllers 104, and if necessary with the ADC device platforms 105. The upgrade utility 101 then contacts the host 103 and uses resources on the host 103 to communicate with the controllers 104. For example, the host 103 may identify the type of the transportation network 107 to the upgrade utility 101. The upgrade utility 101 examines the controller 104 for its characteristics pertinent to the upgrade. Alternatively, the upgrade utility 101 may consult a database 106 that contains the characteristics of the controllers 104 and the ADC device platforms 105. The upgrade utility 101 then locates the appropriate upgrade, determines how to proceed with the upgrade given the controller's characteristics, and

provides the controller 104 with the necessary upgrade. If the upgrade also applies to the ADC device platforms 105, then the upgrade utility 101 examines the ADC device platform 105 for its characteristics pertinent to the upgrade. Alternatively, the upgrade utility 101 may consult the database 106 for pertinent ADC device platform characteristics. The upgrade utility 101 then locates the appropriate upgrade, determines how to proceed with the upgrade given the ADC device platform's characteristics, and provides the ADC device platform 105 with the necessary upgrade. Finally, the upgrade utility 101 ensures that the controllers 104 and the ADC device platforms are returned to the same state they were in prior to the upgrade.

1.1. DCS Upgrade Utility

1.1.1. Purpose and use of feature

The DCS Upgrade Utility is a tool to manage the distribution of upgrade files placed on the Data Collection Server 300 (DCS 300). The DCS upgrade utility will allow the upgrade process to be started from the DCS 300, or from the ESD tool. The ESD tool can be at a remote site (remotely) or on the target DCS 300 (locally). The DCS upgrade utility will eliminate diskette swapping and prompting.

BIOS upgrades will continue to be accomplished by diskette.

There are four classifications of upgrades: Minor, Reboot, Shutdown, and Severe. A minor upgrade does not require rebooting or shutting processes down. A major C upgrade requires the DCS 300 to reboot. A major D upgrade requires some processes like data collection to shutdown. When the upgrade is finished, the stopped processes will be restarted. A severe upgrade is an upgrade that requires changing partition sizes and/or changing operating systems. This also requires the DCS 300 to reboot. If a severe upgrade is necessary, it will require an upgrade CD-ROM in the CD-ROM drive.

1.1.2. Results of feature usage

Previous upgrades required disk swapping and considerable user input. Sometimes the upgrade took hours. An operator needed to be present to swap diskettes. This usually meant that during business hours the controller was unavailable for data collection while an upgrade was being accomplished.

With the DCS Upgrade Utility in conjunction with the ESD tool, a DCS 300 software upgrade can be scheduled for an inactive time and doesn't require that an operator be present when the upgrade is being accomplished. The amount of time an upgrade takes to perform is also reduced.

Initiating an upgrade, from the System Maintenance Menu at the DCS 300, by selecting DCS Upgrade Utility, also requires little attention. Just start the upgrade and walk away.

1.1.3. Feature Options

The upgrade can be started remotely or at a DCS 300 using the ESD tool. A start time can be chosen so the upgrade doesn't interfere with data collection. The upgrade can also be started from the System Maintenance Menu at the DCS 300 by selecting DCS Upgrade Utility.

1.1.4. Assumptions

To use the DCS Upgrade Utility, the DCS software must be at 300 ver 1.0 or greater. Or, To use the DCS Upgrade Utility, the controller must be at 0200 ver 3.0 and have the DCS Upgrade Utility installed. In this case, if the upgrade is started from the controller, "g:\upgrade\upgrade" must be entered at the command line. The DCS Upgrade Utility can be installed from ESD or from a diskette.

The upgrade files can be transferred over to the DCS 300 by ESD, or FTP without the use of a CD-ROM. The upgrade files can be transferred from an Intermec Web page to a DCS 300 with ESD. Or, the upgrade files can be transferred from an Intermec Web page to a PC. Then, ESD or FTP in the binary mode can transfer the files to the DCS 300. If the upgrade

files are to reside on the DCS 300, up to 120 MegaBytes must be available on drive d: of the hard drive depending on the size of the upgrade files.

An upgrade CD-ROM must be in the CD-ROM drive or these files need to be present in the d:\upgrade directory to start the upgrade:

FILE	WHEN THE NEW FILE IS COPIED
upgrade.exe	only when the file changes
*.zip	every upgrade

If a CD-ROM is used in the upgrade of an 0200 controller , the user will need to enable the parallel port in the BIOS at the beginning of the upgrade and disable the parallel port in the BIOS at the end of the upgrade. The DCS 0300 comes with a CD-ROM drive installed and no BIOS changes are necessary.

1.1.5. Interfaces to initiate or exercise the feature

1.1.5.1. Making upgrade files available to the DCS 0300

For each upgrade, new upgrade files will need to be made available to the DCS 300 before the upgrade is initiated. There are several methods to make the files available. The user can place a CD-ROM in the CD-ROM drive, or the user can put the files in the d:\upgrade directory with ESD or FTP. There are different ways to make the files available because some users do not have the capability to utilize TCP\IP and won't be able to use ESD remotely or won't be able to use FTP.

1.1.5.1.1. CD_ROM

When using a CD-ROM the user will place an Upgrade CD-ROM in the CD-ROM drive and initiate the upgrade from the System Maintenance Menu or from ESD (remotely of locally).

1.1.5.1.2. ESD Tool

When using the ESD tool remotely, the user will download a file (e.g. 300V1_0.zip) from an Intermec Web page to a PC. The ESD tool will unzip the file in a directory. The ESD tool will copy the upgrade files to the d:\upgrade directory on the target DCS 300. The user will then schedule the upgrade using the ESD tool. ESD will initiate the upgrade remotely at the appropriate time. To use this method, the target DCS 300 must have a LAN card installed with IP enabled.

Alternately, the user can put an upgrade CD-ROM in the CD-ROM drive at the target DCS 300 and schedule the upgrade from ESD (locally or remotely).

1.1.5.1.3. FTP

When using FTP, the user will download a file (e.g. 300V1_0.zip) from an Intermec Web page to a PC. The user will unzip the file in a directory. The user will login into the DCS 300 using FTP and set the binary mode. The user will change directories to d:\upgrade. Then, the user will copy all the files to the DCS 300. To use this method, the target DCS 300 must have a LAN card installed with IP enabled.

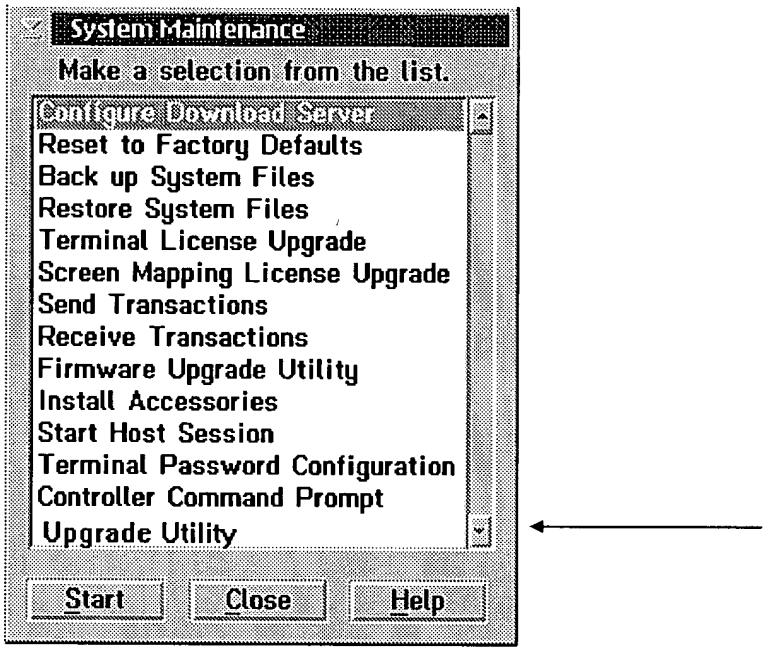
The upgrade can then be started at the DCS 300, from the System Maintenance Menu, or by ESD (remotely or locally).

1.1.5.2. Starting the DCS Upgrade Utility

An upgrade can be initiated from ESD remotely or locally or the upgrade can be started from the DCS 300 System maintenance menu.

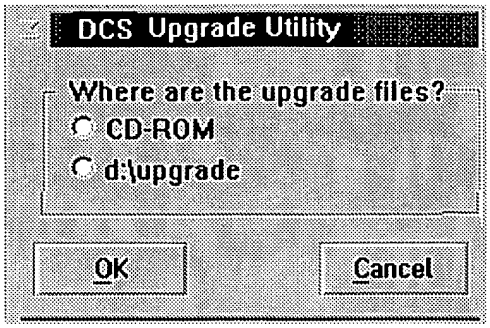
See the ESD section for more information on initiating an upgrade from ESD.

To initiate an upgrade from the DCS 300, select DCS Upgrade Utility from the System Maintenance Menu.

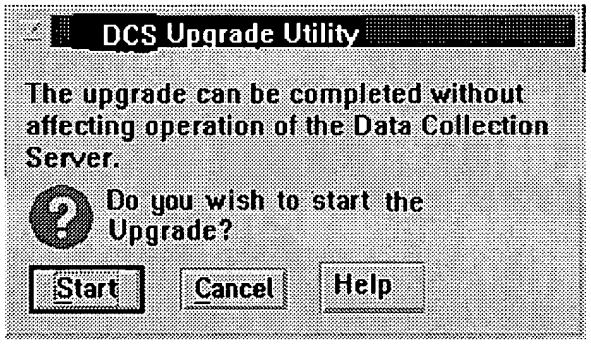


When DCS Upgrade Utility has been selected a box will show that asks the source of the upgrade files. After the source of the upgrade files has been entered a menu will come up and ask if the user wants to start the upgrade or cancel. Pressing start will initiate the upgrade. Pressing cancel with end the upgrade.

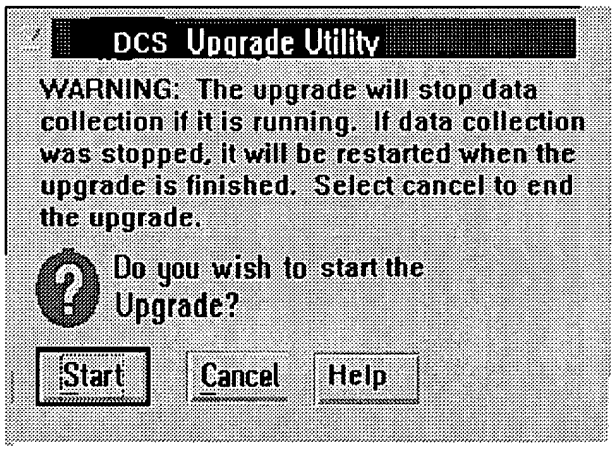
Prompt for source of upgrade files:



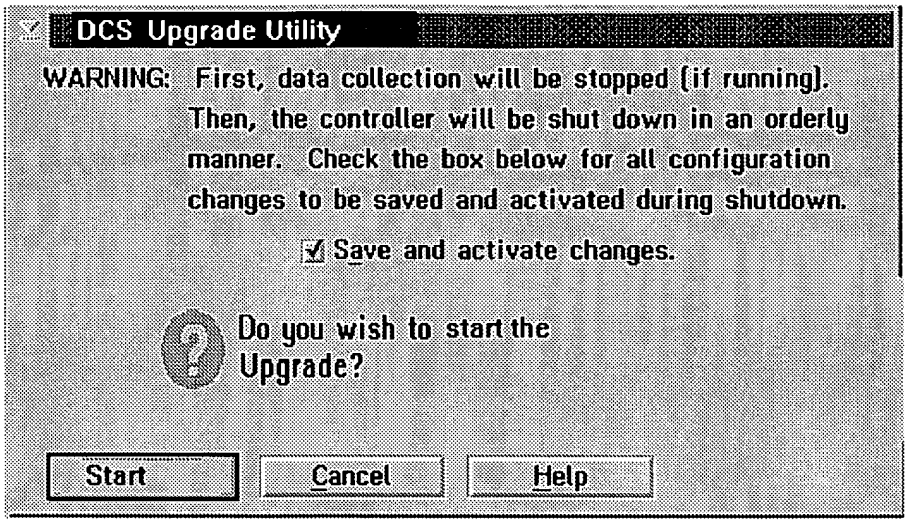
For a minor upgrade, this last chance box will be displayed:



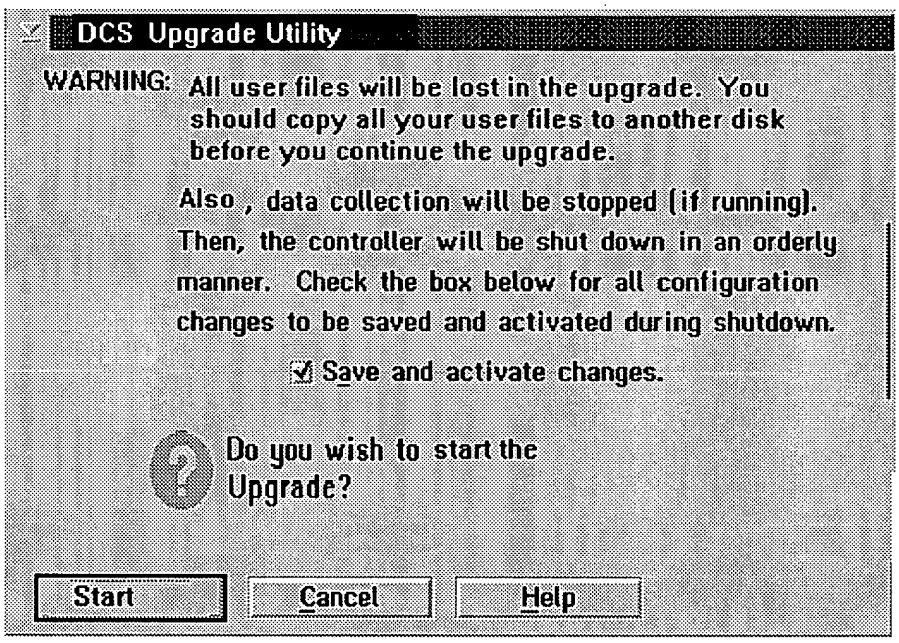
For a major D: upgrade, this last chance box will be displayed:



For a major C: upgrade, this last chance box will be displayed:



For a severe upgrade, this last chance box will be displayed:



Once Start button has been pressed on any of the above mentioned dialogs, the upgrade cannot be stopped.

1.1.5.3. Upgrade Behavior

When the upgrade is started, a file, upgrade.ini, is extracted from the zipped up files. This file contains the parameters of the upgrade, the software version of the upgrade, and a version controller.

The version controller is a list of versions of DCS 300 software that can be upgraded to its upgrade software version. The DCS 300's software version is compared to the

version controller. If the DCS 300's software version is in the list, then the upgrade will proceed. A readme.doc can be found in the same directory as the upgrade files (d:\upgrade or CD-ROM). The readme.doc file contains information on what versions of DCS 300 software the upgrade files can upgrade.

The parameter list in the upgrade.ini file controls the upgrade behavior. It controls if the DCS 300 needs to reboot, shut down various DCS 300 processes, or proceed as is. The rebooting and process control is done automatically and requires no user input.

1.1.5.4. User Interface

A user at the DCS 300 during an upgrade will see various messages at a command prompt window on the DCS 300 screen depending on the size of the upgrade.

If the upgrade is a severe upgrade, the DCS 300 will reboot to an alternate operating system. These are the messages that can show on the screen based on additional upgrade.ini parameters:

- Backing up system files
- Deleting partitions
- Creating partitions
- Formatting partitions
- Restoring system files
- Checking video drivers
- Restoring configuration

In addition to these messages, the user will see a list of files being copied to the appropriate directories after the partitions are formatted.

If the upgrade is a major C upgrade, the DCS 300 will reboot to an alternate operating system. These are the messages that can show on the screen based on additional upgrade.ini parameters:

- Backing up system files
- Restoring system files
- Checking video drivers
- Restoring configuration

In addition to these messages, the user will see a list of files being copied to the appropriate directories after the system files are backed up.

If the upgrade is a major D upgrade, these are the messages that can show on command prompt window on the DCS 300 screen:

- stopping data collection
- starting data collection
- shutting down the DCS 300 GUI
- starting the DCS 300 GUI

In addition to these messages, the user will see a list of files being copied to the appropriate directories after the processes have been shut down.

If the upgrade is a minor upgrade, then all the user will see is a list of files being copied to the appropriate directories.

The ESD tool can also inquire the version of the upgrade software on the DCS 300 and the version of the DCS 300 software to make sure the upgrade will proceed normally.

The DCS upgrade Utility will behave the same when the operating system is ported to Windows NT in a future release of the DCS 300.

1.2. DCS Upgrade Utility

Currently a new upgrade kit is developed for each release. The latest was on CD-ROM. Previous upgrade kits were on diskette and took a long time to install.

With the DCS Upgrade Utility, changes to the C: drive will be zipped up into a file called os_drive.zip. Changes to the D: drive drive will be zipped up into a file called nextgen.zip. And, changes to the F: drive will be zipped up into a file called boot.zip. The upgrade files will either be and placed in the D:\upgrade directory with the ESD tool, from CD-ROM, or with FTP if the upgrade is ran from the d:\upgrade drive. If the upgrade is ran from the CD-ROM, the zip files will need to be on the CD-ROM. The upgrade can then be initiated by selecting DCS Upgrade Utility from the System Maintenance Menu, scheduled and initiated by ESD, or from the command line. The DCS Upgrade Utility will allow zipped-up files, on CD-ROM or placed in the d:\upgrade, to be exploded into the appropriate directories.

1.2.1. Making the zip files

One zip file for each drive will be needed. The zip file will be made with the use of a batch file. The batch file will contain all the names and directories of the files that need to be updated. A typical command in the batch file to zip a single file is "zip -a os_drive.zip c:\ibmcom\1e100t.rsp". This line adds (-a) c:\ibmcom\1e100t.rsp to the zip file os_drive.zip. To zip the entire ibmcom directory, use this command: "zip -a -r -S os_drive.zip c:\ibmcom". Because the files that need updated are different for each upgrade, new zip files will be needed for each

The same upgrade.ini needs to be zipped up in every zip file. For REBOOT and SEVERE upgrades a list of backup files (upgdbkup.lst) will also need to be zipped up in the zip files. This upgrade.ini is used to verify that the zip file is valid. These are the valid zip file names:

- os_drive.zip - These are changes to the c: drive which contains the main operating system.
- nextgen.zip - These are changes to the d: drive which contains the DCS software. The changes to the upgrade.exe should not be in here unless the upgrade is SEVERE. This is because the upgrade.exe on the d:\upgrade directory will be running if the upgrade was initiated from the d:\upgrade drive.
- boot.zip - These are changes to the f: drive which is the alternate operating system used by SEVERE and REBOOT upgrades.

1.2.2. Upgrade Files

This is what the directory structure should look like on the CD-ROM and in the d:\upgrade directory:

```
Volume in drive D is UNOVA
Volume Serial Number is D0AD-8D81

 Directory of D:(OR G:)\UPGRADE\

01/01/01  12:00a     <DIR>          .
01/01/01  12:00a     <DIR>          ..
12/08/97  01:17p        83,418,895 OS_DRIVE.ZIP
12/08/97  01:21p        12,131,654 NEXTGEN.ZIP
```

```
12/08/97  01:21p        12,131,654 BOOT.ZIP
12/08/97  01:21p        12,131,654 UPGRADE.EXE
09/04/97  08:31a       <DIR>          SVGA
09/04/97  08:31a       <DIR>          VGA
09/04/97  08:31a       <DIR>          TOOLS
          7 File(s)     95,550,549 bytes

 Directory of D:(OR G:)\UPGRADE\SVGA

09/04/97  08:31a       <DIR>          .
01/01/01  12:00a       <DIR>          ..
01/22/94  11:29a            23,840 BVHSVGA.DLL
04/08/94  11:29a           123,645 IBMDEV32.DLL
11/30/93  11:29a            38,704 IBMVGA32.DLL
04/09/94  11:29a            77,436 VSVGA.SYS
          4 File(s)         250,549 bytes
Directory of D:(OR G:)\UPGRADE\VGA

09/04/97  08:31a       <DIR>          .
01/01/01  12:00a       <DIR>          ..
08/12/96  11:29a            32,750 BVHSVGA.DLL
08/04/96  11:29a            82,891 IBMDEV32.DLL
07/16/96  11:29a            34,439 IBMVGA32.DLL
08/29/96  11:29a           147,965 VSVGA.SYS
          4 File(s)         250,549 bytes

 Directory of D:(OR G:)\UPGRADE\TOOLS

09/04/97  08:31a       <DIR>          .
01/01/01  12:00a       <DIR>          ..
07/10/97  11:29a             2,149 CARD.IRQ
06/05/97  01:58p           138,291 UNZIP.EXE
8 File(s)      233,628 bytes

 Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG

09/04/97  08:31a       <DIR>          .
09/04/97  08:31a       <DIR>          ..
09/04/97  08:31a       <DIR>          1E
09/04/97  08:31a       <DIR>          1E100
09/04/97  08:31a       <DIR>          1E100E
09/04/97  08:31a       <DIR>          1E100ET
09/04/97  08:31a       <DIR>          1E100T
09/04/97  08:31a       <DIR>          1E1T
09/04/97  08:31a       <DIR>          1T
09/04/97  08:31a       <DIR>          2E
09/04/97  08:31a       <DIR>          2E100
09/04/97  08:31a       <DIR>          2E100T
09/04/97  08:31a       <DIR>          2E1T
09/04/97  08:31a       <DIR>          LAN_LESS
09/04/97  08:31a       <DIR>          LIC_STRT
09/04/97  08:31a       <DIR>          ONE_RF
07/19/96  03:34p                37 STARTUP.CMD
09/04/97  08:31a       <DIR>          TWO_RF
```

09/04/97 08:31a <DIR> T_AX_CM2
19 File(s) 37 bytes

Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\1E

09/04/97 08:31a <DIR> .
09/04/97 08:31a <DIR> ..
06/04/97 08:30a 761 PROTOCOL.INI
3 File(s) 761 bytes

Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\1E100

09/04/97 08:31a <DIR> .
09/04/97 08:31a <DIR> ..
04/30/97 11:15a 786 PROTOCOL.INI
3 File(s) 786 bytes

Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\1E100E

09/04/97 08:31a <DIR> .
09/04/97 08:31a <DIR> ..
06/02/97 02:35p 983 PROTOCOL.INI
3 File(s) 983 bytes

Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\1E100ET

09/04/97 08:31a <DIR> .
09/04/97 08:31a <DIR> ..
06/02/97 02:22p 1,367 PROTOCOL.INI
3 File(s) 1,367 bytes

Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\1E100T

09/04/97 08:31a <DIR> .
09/04/97 08:31a <DIR> ..
04/30/97 11:17a 1,160 PROTOCOL.INI
3 File(s) 1,160 bytes

Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\1E1T

09/04/97 08:31a <DIR> .
09/04/97 08:31a <DIR> ..
06/04/97 08:31a 1,134 PROTOCOL.INI
3 File(s) 1,134 bytes

Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\1T

09/04/97 08:31a <DIR> .
09/04/97 08:31a <DIR> ..
05/01/96 02:07p 938 PROTOCOL.INI
3 File(s) 938 bytes

Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\2E

```
09/04/97  08:31a      <DIR>          .
09/04/97  08:31a      <DIR>          ..
06/04/97  08:31a                 962 PROTOCOL.INI
               3 File(s)        962 bytes

 Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\2E100

09/04/97  08:31a      <DIR>          .
09/04/97  08:31a      <DIR>          ..
04/30/97  11:18a               1,013 PROTOCOL.INI
               3 File(s)      1,013 bytes

 Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\2E100T

09/04/97  08:31a      <DIR>          .
09/04/97  08:31a      <DIR>          ..
04/30/97  11:19a               1,393 PROTOCOL.INI
               3 File(s)      1,393 bytes

 Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\2E1T

09/04/97  08:31a      <DIR>          .
09/04/97  08:31a      <DIR>          ..
06/04/97  08:31a               1,342 PROTOCOL.INI
               3 File(s)      1,342 bytes

 Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\LAN_LESS

09/04/97  08:31a      <DIR>          .
09/04/97  08:31a      <DIR>          ..
02/28/96  11:54a                 216 NGSETUP.CMD
10/02/95  10:25a                 627 PROTOCOL.INI
02/28/96  11:54a               1,509 TCPSTART.CMD
               5 File(s)      2,352 bytes

 Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\LIC_STRT

09/04/97  08:31a      <DIR>          .
09/04/97  08:31a      <DIR>          ..
04/04/96  07:41a                  17 DCMF1.LIC
02/08/96  03:13p                  17 DCMF2.LIC
02/08/96  03:15p                  17 DCMF3.LIC
06/20/96  12:54p                  17 DCML1.LIC
04/04/96  07:42a                  17 DCML2.LIC
02/08/96  03:14p                  17 DCML3.LIC
03/21/97  07:38a               1,271 NGSYS.BAK
06/26/97  12:54p               1,256 NGSYS.INI
02/08/96  03:14p                  17 NOLIMIT.LIC
02/08/96  03:15p                  17 NOLIMITF.LIC
              12 File(s)      2,663 bytes

 Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\ONE_RF

09/04/97  08:31a      <DIR>          .
```

```
09/04/97  08:31a      <DIR>          ..
06/02/97  02:07p                 388 NET.CFG
          3 File(s)        388 bytes

 Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\TWO_RF

09/04/97  08:31a      <DIR>          .
09/04/97  08:31a      <DIR>          ..
06/02/97  02:08p                 217 NET.CFG
          3 File(s)        217 bytes

 Directory of D:(OR G:)\UPGRADE\TOOLS\CONFIG\T_AX_CM2

09/04/97  08:31a      <DIR>          .
09/04/97  08:31a      <DIR>          ..
11/01/95  04:43p               4,841 TWAX_CM2.CF2
11/01/95  04:43p              28,928 TWAX_CM2.CFG
11/01/95  04:43p               2,782 TWAX_CM2.NDF
11/01/95  04:43p                 325 TWAX_CM2.SEC
          6 File(s)      36,876 bytes

    Total Files Listed:
         126 File(s)    96,573,903 bytes
```

If an upgrade is to be performed from the d:\upgrade drive, and changes to this directory tree should be copied over before upgrade begins.

1.2.3. Starting the DCS Upgrade Utility

The DCS upgrade utility can be started from ESD, from the System Maintenance Menu, or from the command line. The Upgrade Utility is a stand alone application with the following command line syntax:

upgrade [q or s]

The qualifier "q" is optional. It is used to pre-test the upgrade to see if it will proceed normally. The "s" qualifier is used to indicate that a save and activate will be done if it is necessary. Either the "q" or "s" is used, not both at the same time.

1.2.3.1. ESD

IF ESD is used, ESD will first send a system transaction to the message handler with these one of these sets of parameters:

- "d:\upgrade\upgrade /q" - The query of the upgrade started from the d: drive.
- "g:\upgrade\upgrade /q" - The query of the upgrade started from the CD-ROM

A query will just run the phase 0 part of the upgrade and place the results in a file (upgrade.log). The results include error messages, parameters of the upgrade, the current phase of the upgrade, and the size of the disk drive. ESD will check upgrade.log for error messages. If error messages are found it will not proceed with the upgrade. If

no error messages are found, ESD will send a system transaction to the message handler with these parameters:

- "\f:\upgrade\upgrade" - The upgrade started from the d: drive
- "g:\upgrade\upgrade - The upgrade started from the CD-ROM
- "/f:\upgrade\upgrade /s" - The upgrade started from the d: drive and a save and activate will be done if necessary.
- "/g:\upgrade\ upgrade /s" - The upgrade started from the CD-ROM and a save and activate will be done if necessary

1.2.3.2. System Maintenance Menu

When the upgrade is started from the system maintenance menu, upgrade.exe is started by 300UgradeUtil in ngpblist.c. 300UgradeUtil will prompt the user for the source of the upgrade file, then call upgrade.exe with these parameters:

- "d:\upgrade\upgrade /q" - The query of the upgrade started from the d: drive.
- "g:\upgrade\upgrade /q" - The query of the upgrade started from the CD-ROM

When the query is done 300UgradeUtil will parse upgrade.log for error messages. If error messages are found, a message will be displayed on a message box on the GUI and the upgrade will end. If there was no errors, upgrade.ini will be parsed again for the parameters of the upgrade, and prompt for a last chance cancel or start based on the parameters. If start is selected, 300UgradeUtil will call upgrade.exe with these parameters:

- "\f:\upgrade\upgrade" - The upgrade started from the d: drive
- "g:\upgrade\upgrade - The upgrade started from the CD-ROM
- "/f:\upgrade\upgrade /s" - The upgrade started from the d: drive and a save and activate will be done if necessary.
- "/g:\upgrade\ upgrade /s" - The upgrade started from the CD-ROM and a save and activate will be done if necessary

1.2.3.3. Command Line

When the upgrade is started from the command line, the user will enter send "f:\upgrade\upgrade" or "g:\upgrade\upgrade". This should only be used when a 0200 Controller is upgraded to DCS 300, ver 1.0. No save and activate will be done.

1.2.4. Upgrade control

In phase 0 of the upgrade, upgrade.ini will be extracted from each zip file. This file will contain the type of the upgrade, the subsections of the type of upgrade, the software version of the upgrade, the version level, and a version controller. The parameters will be used to select the sections of the upgrade that need to be performed. As enhancements are made to the DCS 300, and new sections to the upgrade process are identified, additional parameters can be added to this file and the upgrade executable. The upgrade.ini file is also for security. If the file cannot be extracted from a zip file then the upgrade will not proceed. This is what an upgrade.ini will look like:

[Version_Level]

Version = 300 x.x

[Upgrade]

Type = {SEVERE, MINOR, REBOOT, or SHUTDOWN}
Version = DCS 300 x.x

[Changed_Software]

OS = {YES or NO}
NEXTGEN = {YES or NO}
SWAPPER = {YES or NO}
BOOT = {YES or NO}
BOOT_MANAGER = {YES or NO}

[Changed_Partitions]

OS = {YES or NO}
NEXTGEN = {YES or NO}
SWAPPER = {YES or NO}
BOOT = {YES or NO}
BOOT_MANAGER = {YES or NO}

[OS]

START = { BOTTOM or TOP}
File_System = { FAT or HPFS}
Vtype = { PRIMARY or SECONDARY}
540_NAME = { currently 0000003f}
540_SIZE = { the size of the new partition}
2200_NAME = { e.g. 0000003f}
2200_SIZE = { the size of the new partition}
2500_NAME = { currently 0000003f}
2500_SIZE = { the size of the new partition}

[SWAPPER]

START = { BOTTOM or TOP}
File_System = { FAT or HPFS}
Vtype = { PRIMARY or LOGICAL}
540_NAME = { e.g. 0000003f}
540_SIZE = { the size of the new partition}
2200_NAME = { e.g. 0000003f}
2200_SIZE = { the size of the new partition}
2500_NAME = { e.g. 0000003f}
2500_SIZE = { the size of the new partition}

Restore_START = { BOTTOM or TOP}
Restore_ File_System = { FAT or HPFS}
Restore_ Vtype = { PRIMARY or LOGICAL}
Restore_540_SIZE = { the size of the new partition}
Restore_2200_SIZE = { the size of the new partition}
Restore_2500_SIZE = { the size of the new partition}

[NEXTGEN]

START = { BOTTOM or TOP}

File_System = { FAT or HPFS}
Vtype = { PRIMARY or SECONDARY}
540_NAME = { e.g. 0000003f }
540_SIZE = { the size of the new partition}
2200_NAME = {e.g. 0000003f}
2200_SIZE = { the size of the new partition}
2500_NAME = { e.g. 0000003f}
2500_SIZE = { the size of the new partition}

Restore_START = { BOTTOM or TOP}
Restore_ File_System = { FAT or HPFS}
Restore_ Vtype = { PRIMARY or LOGICAL}
Restore_540_SIZE = { the size of the new partition}
Restore_2200_SIZE = { the size of the new partition}
Restore_2500_SIZE = { the size of the new partition}

[BOOT]

START = { BOTTOM or TOP}
File_System = { FAT or HPFS}
Vtype = { PRIMARY or LOGICAL}
540_NAME = { e.g. 0000003f }
540_SIZE = { the size of the new partition}
2200_NAME = { e.g. 0000003f}
2200_SIZE = { the size of the new partition}
2500_NAME = { e.g. 0000003f}
2500_SIZE = { the size of the new partition}

[BOOT_MANAGER]

START = { BOTTOM or TOP}
Vtype = { PRIMARY or LOGICAL}
540_NAME = { e.g. 0000003f }
2200_NAME = { e.g. 0000003f}
2500_NAME = { e.g. 0000003f}

[Processes]
NextgenGui = {YES or NO}
DataCollection = {YES or NO}
FUU = {YES or NO}
ViDaemon = {YES or NO}

[Disk_Size]
Num_Cylinders_540_D_Drive = { e.g. 97}
Num_Cylinders_2200_D_Drive = { e.g. 97}
Num_Cylinders_2500_D_Drive = { e.g. 97}

[VIDEO_FIX]
FIix_Config = {YES or NO}
Copy_Files = {YES or NO}

[Version_Controller]
Range = {0200 Ver 3.0, DCS 300 Ver x.x - y.y, z.z}

1.2.4.1. Version level

The version level refers to the parameters that can be stored in upgrade.ini. Upgrade.exe (Upgrade.c) contains a version level also. This version level refers to the parameters that upgrade.exe can read. The version numbers must match. When upgrade.exe is ran, it will compare the two version numbers. If the numbers don't match, the upgrade will end.

1.2.4.2. Upgrade

The Upgrade header contains the type of upgrade and the version of the upgrade. These are the types of the upgrade (upgrade.ini):

```
        MINOR   /* no process need to be stopped, no rebooting        */
        REBOOT /* a major upgrade of the c drive including operating system
    or and upgrade of running d: drive processes on a DCS
                    300 with OS/2*/
SHUTDOWN /* a major upgrade of the d drive that needs processes
            shut down. This will not be used until the DCS 300
            is migrated to Windows NT. The DCS 300 processes
            should be written with PC hooks to shut down the
            process remotely. */
SEVERE  /* a severe upgrade of the DCS 300 which includes
           repartitioning or a new operating system */
```

The SEVERE and SHUTDOWN upgrades will need to specify additional information (subsections) in the upgrade.ini.

1.2.4.3. Processes

The Processes header contains the processes that need to shut down. SHUTDOWN will need to specify which processes to stop, such as nextgen.exe or data collection. A NO means "don't shutdown processes" and a YES means "shutdown the process". For all other types of upgrades this subheading is ignored.

1.2.4.4. Changed_Software

Changed software is only used for SEVERE and REBOOT upgrades. Changed software contains a list of the partitions that will have software updates. Changed software is used to indicate that the software in a partition has changed. For a SEVERE upgrade, If the partition size or file system will change, then, the corresponding setting here will also need to be set to YES because the software will have to be put back after the partition has been reformated.

1.2.4.5. Changed_Partitions

Changed partitions is only used for SEVERE upgrades. All other types will ignore this information. Changed Partitions contains a list of the partitions that will be changed. If the partition is set to NO, it will not be changed. If it is set to YES, it will be changed. Not all partitions will change. It is possible that only one will change (it didn't change size, it could change the file system perhaps). If the BOOT partition changed, the SWAPPER partition should also be changed. This is so the BOOT partition can change sizes. These will be set to one if data on the partition changed (files) or the partition information changed (size, file system, ...).

1.2.4.6. OS, SWAPPER, NEXTGEN, BOOT, BOOT_MANAGER

These headings are used only if they are set to YES in the Changed_Partitions heading. They are only used by SEVERE upgrades. All other upgrades ignore this information. The disk drive information is set here. There needs to be partition information for each partition the was set to YES in Changed_Partitions.

The START parameter indicates if the partition is to be created at the bottom or top of the free space on the disk. The current file system is HPFS. This will be changed to FAT for Windows NT. The files system type is used by fdisk when the new partition is created and by format. Vtype indicates if the partition is primary or logical. Usually the operating system is on the primary partition and is on the c: drive. The size information is used to create the partition. These numbers must be correct for the different drive sizes (540M, 2.5 Gig, and 2.2 Gig) or fdisk will return an error. The name of the drive is important to delete the partition using fdisk. The name can be found by doing a fdisk /query. This will need to be done for all three sizes of drives (540M, 2.5 Gig, and 2.2 Gig). If other sizes of disk drives are used, the information will have to be added here too. These parameters are for fdisk.

If the name of the partition is not included in the os, swapper, nextgen, or boot sections (used only if the partition was set to YES in Changed_Partitions), then the partition will not be deleted. If the size was no included the partition will not be created. And, if the file system was not specified, the partition will not be formated.

For example, for DCS 300 ver 1.0, the swapper partition will need to be deleted, and recreated at a smaller size because there will be two new partitions. For the swapper partition, since it needs to be deleted, the name of the partition will need to be included. Since it also needs to be created at a smaller size the, the size should also be included as well as the file system. The nextgen and the OS partitions will not be deleted or recreated so the size, file system, and name information for the partitions should not be included. The information on the nextgen and the OS partitions will change, so OS and NEXTGEN should be set to YES under the changed software heading in upgrade.ini. Two new partitions will be created - the boot partition and the boot manager. The names for these should not be included because they didn't exist before.

1.2.4.7. Disk_Size

Disk size is the number of cylinders in the d: partition for each of the disk sizes. If a new hard drive is added, new entries will need to be added here. Currently in the field there are 540 Meg hard drives and 2.5 Gig hard drives. Soon there will be 2.2 Gig hard drives.

1.2.4.8. Video Fix

There are currently three type of video cards out in the field. For SEVERE upgrades and reboot upgrades, config.sys will be changed when creat_ng is ran. Also, a new config.ngc might be copied over. If the files are changed, they will have to be changed to reference the correct video drivers. Fix_Config will need to be set to YES if the config.ngc or config.sys is changed. If operating system is updated or replaced, then the video driver software for the correct card will have to be copied over to the appropriate directories. Copy_Files will need to be set to YES to cause the files to be copied over.

1.2.4.9. Version_Controller

The version controller will contain the range of software that can be upgraded.

For this release it will be 0200 ver 3.0 that can be upgraded. It can be a list or a range such as 300 ver 3.0 - 3.3, 3.5.

1.2.4.10. Example

This is the upgrade.ini that will be used for upgrades from 0200 Ver 3.0 to DCS 300 Ver 1.0:

```
[Version_Level]

        Version = 1.0

[Upgrade]

        Type =  SEVERE
        Version = DCS 300 Ver1.0

[Changed_Software]

        OS  = YES
        NEXTGEN =   YES
        SWAPPER =  YES
        BOOT =  YES

[Changed_Partitions]

        SWAPPER =  YES
        BOOT =  YES
        BOOT_MANAGER =  YES

[SWAPPER]

        START =  BOTTOM
        File_System = HPFS
        Vtype =  LOGICAL
        540_NAME =
        540_SIZE =  87
        2500_NAME =  004242ff
        2500_SIZE = 317

        Restore_START = BOTTOM
        Restore_ File_System = HPFS
        Restore_ Vtype = LOGICAL
        Restore_540_SIZE = 95
        Restore_2200_SIZE = 0
        Restore_2500_SIZE = 325

[BOOT]

        START =
        File_System = HPFS
        Vtype = LOGICAL
        540_SIZE = 6
```

2200_SIZE = 0
2500_SIZE = 6

[BOOT_MANAGER]

START = TOP
Vtype = PRIMARY

[Disk_Size]
Num_Cylinders_540_D_Drive = 97
Num_Cylinders_2500_D_Drive = 740

[VIDEO_FIX]
FIix_Config = YES

[Version_Controller]
Range = 0200 Ver 3.0

The type of this upgrade is SEVERE because the upgrade needs to delete and create partitions. The only partition that will be deleted is the swapper partition. It is being deleted to make room for two more partitions. The two new partitions are the alternate boot partition (BOOT) and boot manager. The swapper partition will be recreated at a smaller size. The OS and nextgen partitions will not be deleted or recreated, but new files will be copied over.

Since the swapper, the boot, and the boot manager partitions will change size, the SWAPPER, BOOT, and BOOT_MANAGER settings under changed_partitions are set to YES.

Data for all the partitions will change, so all the values in the Changed_Software are set to YES.

For the SWAPPER section, all the information for the partition is added except the name and size of the partition for the 2.2 Gig drive because none exist out in the field.

For the BOOT section, the boot partition name is not included because no boot partition was there before so no deletion of the partition will be attempted. The delete sections of the upgrade looks for the name of the partition to delete. If the name is NULL the section will be skipped. The new partition will be created at the top (START = TOP) of the free space on the hard drive.

For the BOOT_MANAGER section, the new partition will be created at the top (START = TOP) of the free space on the hard drive. The name is not included because boot manager was not installed before.

Creat_ng will be ran, so Fix_Config under Video_Fix will be set to YES.

For the Version Controller section, controller with 0200 Ver 3.0 software can be upgraded to DCS 300 Ver 1.0.

1.2.5. Upgrade.log file

Upgrade messages will be sent to the controller screen as well as to a file, upgrade.log. At the beginning of the upgrade, a default upgrade.log will be copied over to d:\upgrade. Phase 0 messages are used by ESD and 300UgradeUtil to determine if the DCS upgrade utility will proceed without errors. This is what the default message file looks like:

[PHASE]

phase =

[DISK_SIZE]

disk_size =

[VIDEO_CARD]

VIDEO_CARD =

[ERROR_MESSAGES]

[SUCCESS_MESSAGES]

If the upgrade is SEVERE and the d: partition is to be deleted and recreated, the upgrade.log will be copied over to the f: drive befor the d: drive is deleted.

1.2.5.1. Phase heading

At the completion of each phase, the next phase to be ran is set here. There are currently six phases (0, 1, 2, 3 ,4, and 5). None of the upgrades use all six phases. The MINOR upgrade only uses phase 0 and phase 5. A REBOOT upgrade uses phase 0, phase 4, and phase 5. A SEVERE upgrade uses phase 0, phase 1, phase 2, phase 3, and phase 5. A SHUTDOWN upgrade only uses phase 0 and phase 5.

1.2.5.2. Disk Size heading

The disk size is written to upgrade.log if the upgrade is SEVERE. An IOCTL command is ran that queries the number of cylinders in drive.

1.2.5.3. Video Card heading

This is used for SEVERE and REBOOT upgrades.

1.2.5.4. Error Messages

These are the error messages that can be put in upgrade.log in phase 0:

- Invalid disk size
- Invalid zip files(s)
- The upgrade files are not compatible with the DCS upgrade utility
- The path for the upgrade files is incorrect.
- The upgrade.ini is incorrect.
- Incorrect number of parameters
- DCS 300 ver x.x can not be upgraded to DCS 300 ver y.y. You must first upgrade to DCS 300 ver z.z the use this upgrade.

These are the error messages that can be put in upgrade.log in all other phases:

- Creat_ng failed. The error messages can be found in **TBD.**
- The upgrade completed with errors
- The upgrade could not complete
- Error Copying files to C: Drive
- Error Copying files to D: Drive

- Error Copying files to F: Drive
- Backup failure:
- Restore failure:
- "Could not open master system file list.
- Restore is in progress...
- Backup is in progress...
- Target directory creation error.
- ERROR - Access to drive denied
- Could not open the migration list file.

1.2.5.5. Success Messages

These are the success messages that can be put in upgrade.log in phase 0:

- DCS 300 ver x.x will be upgraded to DCS 300 version y.y.
- 0200 Controller ver 3.0 will be upgraded to DCS 300 version 1.0.

These are the success messages that can be put in upgrade.log in all other phases:

- Successful upgrade

1.2.6. Save and Activate

If the /s parameter was passed in, a SEVERE and REBOOT upgrade will test to see if a save and activate is necessary. If the save and activate is necessary, the new default files will be copied over. Then data collection will be stopped, CM/2 will be ran, and then LAPs will be initiated. Timers will be used to wait to wait for data collection to stop, and for CM/2 setup and LAPs to finish.

1.2.7. Assumptions

1.2.7.1. Processes that need to be in place for the upgrade utility

It is assumed that there will be a config file that contains information on the software version. This will help control upgrades because the DCS 300's software version can be compared to the version controller in the upgrade files.

For ESD to start the DCS Upgrade Utility, a new functionality needs to be added that can spawn a command line utility based on a transaction.

The GUI needs to be changed to add the DCS 300 Upgrade Utility to the menu options. If the Upgrade Utility was selected, a new procedure in ngpblist.c will start the upgrade with the query option and examine the results in upgrade.log when the query is done. If there were no error in upgrade.log, the GUI will show a last chance box based on the type of upgrade (the type of upgrade is also in upgrade.log). If START was selected, the upgrade utility will be started without the query option.

The GUI needs to be changed to continuously check to see if there is an IPC call for a quiet save and activate. The upgrade utility will need to be able to do a quiet save and avtivate without rebooting or user input. Existing procedures like ActivateCfgOK, DcStopTimer, ActivateAfterStop, CheckCmsetup, ActivateAfterCM2, CheckLapsDone, and ActivateAfterLaps and the procedures they call will need to able to accept two more input parmater (quiet, or not quiet, and no reboot, or reboot possible).

1.2.7.2. Limitations

When an upgrade to DCS 300 ver 1 from 0200 ver 3.0 is done, no save and activate will occur because the changes to the DCS software are not in place. Also, the upgrade will need to be started from the command line for the same reasons.

The SHUTDOWN type of upgrade will not be used until the DCS 300 is migrated to Windows NT. The new DCS 300 processes should be written with IPC hooks to stop them remotely.

Much of the previous upgrade kit (upgrade to V3.0 on CD-ROM) can be used. Changes will need to be made to the sections, though. Phases of the upgrade will be used again with the addition of the upgrade parameter list in upgrade.ini.

1.2.8. Software Structure

1.2.8.1. ngpblist.c

For upgrades started from the GUI, code will need to be added to ngpblist.c to add the DCS 300 Upgrade Utility. A new dialog will be created when this option is selected. Under the create event of this dialog a new function (300UgradeUtil) will be added that will prompt the user for which directory the upgrade files are in. Based upon the choice, 300UgradeUtil will call upgrade.exe with these parameters: "upgrade /d:\upgrade /q", if "D: Drive" was selected or "upgrade /g:\upgrade /q" if the "CD-ROM" option was selected. Upgrade.exe will perform phase 0 of the upgrade and quit. Phase 0 determines if the upgrade files are valid, extracts upgrade.ini from the upgrade files, determines if the upgrade.exe is compatible with the upgrade.ini, and determines the type of upgrade. The results of phase 0 is output to upgrade.log. Then upgrade.exe ends. The CreateUpgradeUtil function spawns the upgrade.exe. It then starts a xvt_timer and for each timer event generated, calls CkUpgradeLogFile. This function looks for the update.log file and when successfully opened verifies that no errors were generated by the upgrade.exe application. If errors are present, they are presented to the user in a XVT message dialog otherwise the success message that was written to the log file is displayed. If there were no errors, and based on the type of upgrade, a last resort box will show asking if the user wants to start or cancel the upgrade.

This is the procedure that SideButtonsMenuListOk call for the upgrade utility:

- SideButtonsMenuListOK
- CreateUgradeUtil

1.2.8.1.1. Pseudo-code for ngpblist.c

```
/*****************************************************************************
*
* NAME:        SideButtonsMenuListOK
*
* DESCRIPTION:  Handles dialog OK processing for the
*               Menu List dialog. The dialog is called for the
*               item selected in the list.
*
* REVISION:
* 09/22/95  BK  Added (LONG) pItem to xvt_dlg_create_res call
```

```
* 09/29/95  SL  Added NgStatusMonitor call
* 10/16/95  SL  Remove NgStatusMonitor call
* 10/25/95  SL  Added NgStatusMonitor call
* 11/09/95  SL  Added DisplayErrorlog call
* 11/12/95  SL  CreateAppListDlg call
* 11/12/95  SL  CreateSendTranDlg call
* 11/15/95  SL  CreateTraceDlg call
* 12/19/96  BK  Destroy menu list dlg before calling selected function.
*           Can't create a modeless dlg from a modal dlg.
* 11/6/97   DH  Added DCS upgrade utility entry
*****************************************************************************/
VOID SBMenuListOK (WINDOW xdWindow)
                  /* dialog window handle   */
{
  PSBMENULIST psbMenuList;  /* menu list structure passed in        */
  PSBMENUITEM pItem;        /* menu item from list                  */
  INT         iIndex;       /* index in listbox of item selected    */

  psbMenuList = (PSBMENULIST)xvt_vobj_get_data (xdWindow);

  iIndex = xvt_list_get_sel_index (CTL_WIN(LB_AVAILABLE));
  pItem = psbMenuList->psbMenuItems + iIndex;


  if (pItem->iRid == DB_STATUSMONITOR)
    NgStatusMonitor ();
  else if (pItem->iRid == DB_ERRORLOG)
    DisplayErrorlog ();
  else if (pItem->iRid == DB_APPLICATION)
    CreateAppListDlg ();
  else if (pItem->iRid == DB_TRACE)
    CreateTraceDlg ();
  else if (pItem->iRid == DB_UPGRADEUTILITY)
    UpgradeUtility (FUU_START_GUI);
  else if (pItem->iRid == DB_SENDTRAN)
    CreateSendTranDlg ();
else if (pItem->iRid == DB_300_UPG_UTIL)
   /* the DCS upgrade utility was selected */
    CreateUgradeUtil ();

  else if (pItem->eh != 0)
  {
    if (!xvt_dlg_create_res (WD_MODAL, pItem->iRid, EM_ALL,
                      pItem->eh, (LONG) pItem))
      xvt_dm_post_error ("Can't open dialog");
  }
  else
  {
    xvt_dm_post_error ("Not implemented at this time.");
  }

  return;
}
```

```
/*****************************************************************************
*
* NAME: CreateUpgradeUtil
*
* DESCRIPTION:  Ask user where the upgrade files are located.
*               Determines the type of upgrade: sever, minor,
*               reboot, or  shutdown.  Based on the type of upgrade,
*               present a last chance to quit box
*
* ASSUMPTIONS:  None.
*
*----------------------------------------------------------------------------
*
* REVISION HISTORY:
*
*   Date      Author      Description
*   ----------------------------------------------------------------------
*   11-6-97   Doug Hughes    Original code.
*
*****************************************************************************/

VOID  CreateUpgradeUtil (WINDOW xdWindow)
                    /* Dialog box window handle.  */
{
                    Show a  dialog box that asks where the upgrade files are located.  Show 2 choices,
                    CD-ROM and "D: Drive".  Make the D drive the default.
                    If the choice made was Cancel, quit the upgrade.
                    Check to see if save and activate was unselected.
                    If the choice made was Start, continue
                    call ugprade.exe with "upgrade /g:\upgrade /s" or "upgrade /g:\upgrade /s" if save
                    and activate was selected.
                    call ugprade.exe with "upgrade /g:\upgrade" or "upgrade /g:\upgrade" if save and
                    activate was  unselected.

}
```

1.2.8.2. UpgradeUtilOk (WINDOW xdWindow)

This function is called when the user clicks the OK button in the DCS Upgrade Utility dialog.
If CD-ROM was selected as the source then verify a CD is in the drive.
If not display an error.
Spawn the upgrade.exe application using the /q (query) option for a parameter.
Start the xvt_timer

1.2.8.3. CkUpgradeLogFile(VOID)

This function is called when a E_TIMER event is received in the DCS Upgrade Utility dialog.
Loop until a successful open is performed on the upgrade.log file or until nn timer events have be generated. If we time out, then inform the user and quit the process.
After a successful open is performed, parse the upgrade.log file using the function:

```
LONG FindValue ( PSZ  pszFilePath,
                 PSZ  pszSection,   /* "[ERROR_MESSAGES] */
                 PCHAR szFindString)
```

- If no errors are found then parse the file again and display the messages written under the "[SUCCESS_MESSAGES]" section in a xvt_note dialog.
- When we return from the note dialog, parse the upgrade.log file and find the type, (section = "[UPGRADE_TYPE]").
- Based on the type, display the appropriate dialog (see functional spec. for the different levels of upgrades.

  If the type is MINOR or SHUTDOWN, show the user a Start or Cancel type dialog allowing the user to proceed or abort the process. Upgrade.exe is again called with no paramters.

  IF the type is REBOOT or SEVERE, show the user a Star or Canel type dialog along with a check box to Save and Activate their current configuration.

Upgrade.exe is again called and if the Save and Activate check box is checked, then /s is passed in as a parameter to the call.

1.2.8.4. Upgrade.exe

There are six phases to the upgrade. Phase 0 tests to see if the upgrade files are valid and examines the parameters of the upgrade. Phases 1, 2, and 3 are for SEVERE upgrades only. Phases 1, 2, and 3 delete, create, and format partitions. Phase 4 reboots the DCS 300 to the f: drive if the type of upgrade is REBOOT. MINOR and SHUTDOWN upgrades will jump from phase 0 to phase 5. Phase 5 extracts the zip files.

1.2.8.4.1. Phase 0

Phase 0 does these tests for each possible zip file:

- A check is done to see if the zip file exists (it is not an error if it does not exist)
- If it exists the upgrade.ini is extracted from it
- If there wasn't an upgrade.ini and error is logged to upgrade.log and the upgrade ends.
- If there was an upgrade.inii, its version is compared to the version of upgrade.exe.
- If the versions don't match, an error message is logged in upgrade.log and the upgrade ends.

At this point, if there is no upgrade.ini, an error message is logged to upgrade.log and the upgrade ends. If there is an upgrade.ini the version of the DCS 300 is compared to the version controller in upgrade.ini. If the DCS version doesn't fall in the range in the version controller, an error message is logged to upgrade.log and the upgrade ends. Next the type of upgrade is retrieved and logged to upgrade.log. If it is a SEVERE upgrade, the disk size is also logged to upgrade.log. If no errors have occurred, the success messages is logged to upgrade.log. The message tells the current version of the DCS 300 and the version it is being upgraded to. If the "q" parameter was used to call upgrade.exe, the upgrade will end. If the 'q" parameter was not used the phase is set to the proper value and the upgrade continues. The new phase is logged in upgrade.log.

1.2.8.4.2. Phase 1 (used by SEVERE only)

The DCS 300 will do a save and activate if the /s parameter was passed in.

Phase 1 will extract upgdbkup.lst from one of the zip files and back up the users system files to the d:\upgrade directory. In phase 2 these files will be copied to the boot partition (f: drive).

If BOOT = YES was set in upgrade.ini, phase 1 will delete the swapper partition if the name was set in upgrade.ini. This is so the boot partition can change sizes. Next, the boot partition will be deleted if the the name was set in upgrade.ini. If the delete of the boot partition failed an error message recorded to upgrade.log. The old partition settings will be in the upgrade.ini. If the partitions were deleted successfully, the new boot partition will be created if the size is not equal to zero.

Phase 2 will be entered in the upgrade.log so that when the DCS reboots, phase two will be started.

A new config.sys and startup.cmd will be copied over the old so that when the DCS 300 reboots to the c: drive, the upgrade program will run.

Now the DCS 300 will reboot if it hasn't already.

1.2.8.4.3. Phase 2 (used by SEVERE only)

Phase two will format the swapper partition in the specified file system if the file system was set in upgrade.ini. If it fails, an error message will be recorded to upgrade.log and a error message will print to the screen and the upgrade will end. The DCS 300 is not functional but it is bootable to the C: drive.

If no error occur, the boot partition will be formatted in the specified file system if the file system was set in upgrade.ini. If an error occurs, an error message will be written to upgrade.log and the DCS 300 will be set up to boot up normally and the DCS will reboot to the C: drive. The DCS 300 is fully functional, but the upgrade failed. If no errors occur, boot_drive.zip will be exploded into the new drive and the backed up system files will be copied over as well as the upgrade.ini and upgrade.log files.

If errors occur during the copying, an error message will be written to upgrade.log and the DCS 300 will be set up to boot up normally and the DCS will reboot. The DCS 300 is fully functional, but the upgrade failed. If the e: drive was not deleted before, it can be now if SWAPPER = YES under the changed partitions heading in upgrade.ini. If it fails, an error message will be recorded to upgrade.log and a error message will print to the screen and the upgrade will end. The DCS 300 is fully functional.

If there was no errors and If NEXTGEN = YES under the changed partitions heading in upgrade.ini, the d: drive will now be deleted if the name was set in upgrade.ini. If it fails, an error message will be recorded to upgrade.log, the swapper partition will be recreated, an error message will print to the screen, and the upgrade will end. The DCS 300 is not functional but it is bootable to the C: drive.

If there was no errors and If OS = YES under the changed partitions heading in upgrade.ini, the c: drive will now be deleted if the was set in upgrade.ini. If it fails, an error message will be recorded to upgrade.log, an error message will print to the screen, and the upgrade will end. The DCS 300 is not functional but it is bootable to the c: drive.

If there were no errors, the deleted partitions will now be recreated through fdisk if the sizes were set in upgrade.ini. If a partition creation fails, an error message will be recorded to upgrade.log, an error message will print to the screen, and the upgrade will end. The DCS 300 is not functional and it is not bootable to the c: drive.

If no errors occur creating the partitions, the DCS will reboot to the f: drive if OS, NEXTGEN, or SWAPPER is set to YES under the changed partitions heading in upgrade.ini.

1.2.8.4.4. Phase 3 (used by SEVERE only)

Phase three will first format the os partition in the specified file system if the file system was set in upgrade.ini. If the format fails, an error message will be recorded to upgrade.log, an error message will print to the screen, and the upgrade will end. The DCS 300 is not functional and it is not bootable to the c: drive because there is no operating system on the c: drive. The DCS 300 was booted from the f: drive and is at a command prompt.

If there are no errors, the nextgen partition will be formatted in the specified file system if the file system was set in upgrade.ini. If the format fails, an error message will be recorded to upgrade.log, an error message will print to the screen, and the upgrade will end. The DCS 300 is not functional and it is not bootable to the c: drive because there is no operating system on the c: drive. The DCS 300 was booted from the f: drive and is at a command prompt.

If there are no errors, the swapper partition will be formatted in the specified file system if the file system was set in upgrade.ini. If the format fails, an error message will be recorded to upgrade.log, an error message will print to the screen, and the upgrade will end. The DCS 300 is not functional and it is not bootable to the c: drive because there is no operating system on the c: drive. The DCS 300 was booted from the f: drive and is at a command prompt. If there are no errors, the upgrade zip files are now ready to be exploded - increment the phase to 5 .

1.2.8.4.5. Phase 4 (used by REBOOT only)

The phase is set to 5 in the log file. The DCS 300 will do a save and activate if the /s parameter was passed in and if a save and activate is necessary. Now the DCS 300 will reboot.

1.2.8.4.6. Phase 5

If REBOOT was set in upgrade.ini, upgdbkup.lst will be extracted from one of the zip files. This is the list of files to be backed up and restored. The system files will be copied to the d:\upgrade\backup directory. If the backup fails, an error message will be recorded to upgrade.log, an error message will print to the screen, and the upgrade will reboot the DCS 300 to the c: drive. The DCS 300 is fully functional and operation will resume as normal except the upgrade did not complete successfully.

If the REBOOT parameter is set under the changed partitions heading in upgrade.ini, upgrade.exe will run CreatUpgradeTmpFile. This procedure will read the last successful section from ng_confg.log and copy it to a temp file to be

used when creat_ng is ran. If the CreatUpgradeTmpFile fails, an error message will be recorded to upgrade.log, an error message will print to the screen, and the upgrade will reboot the DCS 300 to the c: drive. The DCS 300 is fully functional and operation will resume as normal except the upgrade did not complete successfully.

If the SHUTDOWN parameter is set under the changed partitions heading in upgrade.ini, the nextgen processes will now be stopped.

For all upgrades, the zip files will now be extracted to the appropriate drive and directories. If OS=YES under the changed_software heading in upgrade.ini, os_drive.zip will be exploded in the c: drive. If the unzip fails, an error message will be recorded to upgrade.log, an error message will print to the screen, and the upgrade will end. The DCS 300 is might or might not be functional and it might or might not be bootable. The status is unknown because the c: drive is corrupted. The DCS 300 was booted from the f: drive and is at a command prompt.

If there is no errors and NEXTGEN=YES under the changed_software heading in upgrade.ini, nextgen_drive.zip will be exploded in the d: drive. If the unzip fails, an error message will be recorded to upgrade.log, an error message will print to the screen, and the upgrade will end. The DCS 300 is might or might not be functional but it is bootable to the c: drive. The DCS 300 was booted from the f: drive and is at a command prompt.

If there is no errors and the REBOOT or SEVERE parameter is set, creat_ng will be ran to restore the user configuration. If the creat_ng fails, an error message will be recorded to upgrade.log, an error message will print to the screen, and the upgrade will reboot to the c: drive and end. The DCS 300 is functional but not configured.

If there is no errors and the REBOOT or SEVERE parameter is set, the procedure systemRestore will be ran to restore the backed up system files. If the systemRestore fails, an error message will be recorded to upgrade.log, an error message will print to the screen, and the upgrade will reboot to the c: drive and end. The DCS 300 is functional but not configured fully.

If there is no errors and the REBOOT or SEVERE parameter is set, the procedure EnableTcpipInConfigSys will be ran to start TCP/IP in the config.sys if it was set before. If TcpipInConfigSys fails, an error message will be recorded to upgrade.log, an error message will print to the screen, and the upgrade will reboot to the c: drive and end. The DCS 300 is functional but not configured fully.

If there is no errors and the REBOOT or SEVERE parameter is set, the procedure HostNAMeRestore will be ran to restore the host name in config.sys. If HostNAMeRestore fails, an error message will be recorded to upgrade.log, an error message will print to the screen, and the upgrade will reboot to the c: drive and end. The DCS 300 is functional but not configured fully.

If the REBOOT or SEVERE parameter is set, the DCS 300 will now reboot to the c: drive and be ready for normal operation.

If the SHUTDOWN parameter is set, the stopped processes will now be restarted.

If there was any errors configuring the controller fully, the user will have to reconfigure the DCS 300.

The main body of upgrade.c calls these functions:

- main
  - SearchAndAddStringtoFile
  - FormatPartition
  - DiskSize
  - CreatePartitions
  - IncrementPhase
  - HostNameRestore
  - FindValue
  - UpdtConfigLog
  - SystemBackup
  - SystemRestore
  - CreateUpgradeTmpFile
  - DosExecPgmWrapper
  - EnableTcpipInConfigSys
  - SaveAndActivate
  - RestoreVideoType()
  - SaveVideoType()

1.2.8.4.7. code for upgrade.c

1.2.8.4.7.1. Main body

Main body

```
/*****************************************************************
***************
*
* FILE NAME:  upgrade.c
*                                                                 *
* PURPOSE:    Main module for upgrade.c, used to do field        *
*                  upgrades. *
* AUTHOR:  D. Hughes, D  Kaatz                                    *
*
* DATE:     11/2/97                                               *
* REVISION HISTORY:                                               *
*    Date       Author       Description                         *
*-------------------------------------------------------------------------
*   11/2/97   Doug Hughes   Original code                         *
*             Dave Kaatz                                          *
* COPYRIGHT (c)1997 INTERMEC CORPORATION, ALL RIGHTS
* RESERVED                                                        *
*****************************************************************
***************/

#define INCL_DOSSESMGR
#define INCL_DOSFILEMGR
#define INCL_DOSPROCESS

#define INCL_DOSDEVICES
#define INCL_DOSDEVIOCTL
#define INCL_DOSFILEMGR
#define INCL_DOSERRORS
#define INCL_DOSPROCESS
#define INCL_DOS

#include <stdlib.h>
#include <string.h>
#include <conio.h>

#include <ctype.h>
#include <locale.h>
#include <fcntl.h>
#include <errno.h>
#include <io.h>

#include "dcsys.h"
#include "dcmuser.h"
#include "lnklstgp.h"
#include "files_gp.h"
#include "libutlgp.h"
#include "autcongp.h"
#include "mem.h"
```

```
#include "creat_ng.h"
#include "field_ng.h"
#include "upgrade.h"

/******
 ****** #DEFINES
 ******/

#define  STR_WARN_TWO "Make a new system backup diskette when the
upgrade has completed.\n"
#define  PROTO_TARGET_FILE          "protocol.tmp"
#define  BACK_PROTO_FILE           "protocol.bac"
#define  OLD_PROTO_TOKENR_SECTION    "[IBMTOK_nif]"
#define  NEW_PROTO_TOKENR_SECTION    "[NGTRING_nif]"

/*
** Global variables
*/
CHAR pszSystemFiles[MAXPATH];
static CHAR szWhites[] = {SP,'\t',NUL};  /* select ignored white chars    */
static CHAR szSkips[] = {SP,'=','\t',NUL};
BOOL bDebug = FALSE;

/*
** Prototypes


LONG DeletePartitions ( PCHAR pszDiskName);
LONG SearchAndAddStringtoFile(PSZ pszFileNamePath, PSZ pszSection,
                              PSZ pszSetting, PSZ pszInsertString)
LONG FormatPartition (PCHAR pszDrive, PCHAR pszFileSystem);
LONG CreatePartitions( LONG lDiskSize,  LONG lType, INT iStartable,
                       PSZCHAR pszFileSystem,  CHAR cStart);

LONG IncrementPhase( VOID );
LONG HostNameRestore( void );
LONG DiskSize(VOID);
SZ FindValue (PSZ pszFilePath, PSZ pszSection,  PSZ  szSetting);
LONG UpdtConfigLog( void );
LONG SystemBackup(PSZ pszPath );
LONG SystemRestore( PSZ pszPath );
LONG CreateUpgradeTmpFile(PSZ pszPath );
LONG DosExecPgmWrapper(PCHAR szPgm, PCHAR szArgs);
LONG EnableTcpipInConfigSys(VOID);
VOID Phase_Zero();
VOID Phase_One();
VOID Phase_Two();
VOID Phase_Three();
VOID Phase_Four();
VOID Phase_Five();
CopyVideoDrivers()
FixVideoInConfigSys()
```

```
/*****************************************************************
***
 main program function

 DESCRIPTION:    The upgrade works in states, called phases here.
          The number of phases is five. The first phase ( phase 0) checks
           to see if the zip files exist and are valid and checks version
           numbers for an appropriate upgrade. Phases 1 thru 3 are for
           SEVERE upgrades. In phases 1 thru 3, the boot partition can
          be deleted, a new created in a specified file system and size.
          Then, the other partitions can be deleted and re-created in the
          specified file system and size. The sixth phase   (phase 5)
           performs the upgrade.

           Current phase is written to upgrade.log on the d:\upgrade
           directory.
          The phase  is updated at the completion of each phase.

           If the upgrade is SEVERE, upgrade.log is copied to the f: drive
           if/after f: has been formated.

 AUTHOR:      Doug Hughes (major) and David Kaatz (assistance)

 DATE:        11/2/97
*****************************************************************
***/
int main(argc argv)

{
  CHAR   szFileNamePath[MAXPATH];
  CHAR   szSearchString[MAXPATH];
  CHAR   szInsertString[MAXPATH];
  INT    iInstance = 0;
  LONG   lDiskSize = 0;
  LONG   lPhase = 0;
  LONG   lRc = RC_OK;
  INT i;
  ULONG  ulReserved;
  CHAR   szPhaseNumber[35];
  CHAR   *pszPhaseNumber;
  UCHAR LoadError[CCHMAXPATH];
  PSZ Envs = NULL;
  PSZ Args = NULL;
  RESULTCODES ChildRC;
  APIRET rc = NO_ERROR;
  LONG lRC = NO_ERROR;
  BOOL  fFileExists;
  BOOL fQuery = FALSE;
  BOOL fSaveActivate = FALSE;
  PSZ pszTemp[NG_BUF_IN_LEN];
  PSZ pszPath[PATH];
  PSZ pszFullPath[PATH];

  BOOL  fHiddenFile;
```

```
PSZ pszUpgradeSource[15};
CHAR  chParameter = NULL;
INT iNumParams = 0;
INT iUpgradeType;
PSZ  pszUpgradeType;

BOOL fBootPartitionChanged = 0;
PSZ  pszBootPartitionName;
LONG lBootPartitionSize;
PSZ pszBootPartitionFS;
CHAR cBootPartitionStart;
 INT  iBootPartitionType;
 BOOL fDeleteAdjancentPartition = 0;
 LONG lAdjacentPartitionName;
 INT iBootPartitionStartable = 0;

BOOL fOSPartitionChanged = 0;
PSZ  pszOSPartitionName;
LONG lOSPartitionSize;
PSZ pszOStPartitionFS;
CHAR cOSPartitionStart;
 INT  iOSPartitionType;
 INT iOSPartitionStartable = 0;

BOOL fNextGenPartitionChanged = 0;
PSZ  pszNextGenPartitionName;
LONG lNextGenPartitionSize;
PSZ pszNextGentPartitionFS;
CHAR cNextGenPartitionStart;
 INT  iNextGenPartionType;
INT iNextGenPartitionStartable = 0;

BOOL fSwapperPartitionChanged = 0;
PSZ  pszSwapperPartitionName;
LONG lSwapperPartitionSize;
PSZ pszSwappertPartitionFS;
CHAR cSwapperPartitionStart;
 INT  iSwapperPartitionType;
INT iSwapperPartitionStartable = 0;


CHAR  szSysCmdStr[NG_BUF_IN_LEN + NG_BUF_IN_LEN +2];  /* for
                                                system copy command  */
CHAR  szErrorOut[MAX_ERROR_OUT_LEN + 1];  /* for error output
                                                  strings   */
INT   iSysReturn = 0;                /* for return from system call */

INT  iIniLevel = 0; /*    the level that upgrade.ini is at. If new
                          parameters are added to upgrade.ini
                          its level would change to 1.1.  This
                          level needs to match iExeLevel.
                         */
INT  iExeiLevel = 1.0; /* the level that upgradeexe is at. If new
                          parameters are added to upgrade.exe
```

```
                its level would change to 1.1.  This
                level needs to match the level in upgrade.ini
              */

  check for upgrade.ini
  if upgrade.ini exists delete it

  Copy generic upgrade.log over to d:\upgrade

  memset(pszPath,NUL,PATH+1);
  memset(pszFullPath,NUL,PATH+1);

  iNumParams = argc;
  if(iNumParams !> 2)
  {
    /* Source of upgrade files not specified */
    print error message to screen and to upgrade log file
    quit upgrade;
  }
  else
  {
     for(i=0; i < iNumParams; i++)
     {
       if(strncmp(StrToUupr(argv(i)),'/Q', 2));
       {
           fQuery = TRUE;
       }
       else if(strncmp(StrToUupr(argv(i)),'/S', 2));
       {
           fSaveActivate = TRUE;
       }
       else if (strstr(argv(i),':\"))
       {
         strcpy(pszPath,argv(i));
       }
     } /* end of for */
   }  /* end of else */
}

  sprintf(pszFullPath,"%s%s",pszPath, LOG_FILE);
  lPhase = FindValue(pszFullPath, PHASE, PHASE);   /* determine the
                                                      phase number */
 memset(pszFullPath,NUL,PATH+1);

/* Phase zero is used by all phases.
   Phase zero checks to see if the zip files are
   valid
*/
if(lPhase == 0)
 {
    Phase_Zero();
 }
/* Phase one is used by SEVERE upgrades only.
   Phase one can delete/create  the boot partition,
```

```
  it can delete/create the boot manager too.
*/
if(lPhase == 1)
 /* severe upgrade */
 {
   Phase_One();
  }

/* Phase two is for SEVERE upgrades only.
          Phase 2 does the following:
          Formats the new boot partition,
          Copies the new operating system to the boot partition
          Delete and create any other partitions
 */
 if(lPhase == 2)
 {
    Phase_Two();
 } /* end of phase 2 */

 /* Phase three is for SEVERE upgrades only.
   Phase three reformats the recreated partitions
 */
 if(phase == 3)
 {
    Phase_Three();
 }

  /* Phase four is for REBOOT upgrades only
    Phase four does a save and activate and
    reboots to the alternate operating system
  */
  If(lPhase == 4)
  /* REBOOT */
  {
     Phase_Four();

   }; /* end of phase 4 */


/* Phase 5 does the following:
        system backup                 REBOOT
        CreateUpgradeTmpFile          REBOOT
        Stop nextgen processes        SHUTDOWN
        explodes *.zip                ALL TYPES
        system restore                REBOOT
        hostname restore              REBOOT
        proto ini merge               REBOOT
        Restart shutdown processes SHUTDOWN
 */

  If(lPhase == 5)
  /* Now booted from operating system on f: if REBOOT or SEVERE */
  {
     Phase_Five();
```

```
  }
} // end of main function
```

1.2.8.4.7.1.1. SaveAndActivate

```
/*****************************************************************
*********
  FUNCTION:    SaveAndActivate
  AUTHOR:      D. Hughes
  DESCRIPTION: Saves and activates coniguration changes, shuts down
               data collection, runs CM/2 settup, and runs LAPS.
               Shuting down data collection, running CM/2 settup, and
               LAPS require seperate timers.

  RETURNS:     RC_OK if sucessfull, RC_ERROR if not sucessful
               -1 otherwise

 REVISIONS:
  DATE   NAME    DESCRIPTION
  ----------------------------------------------------------------
 12/01/97 D.Hughes    INITIAL REVISION

*****************************************************************
*********/
/*
SaveAndActivate()
*/
LONG SaveAndActivate( VOID )
{
  copy new default files.

  initiate stop data collection.
  While(not stopped || timed out)
  {
    sleep 1 second
    counter ++
    test to see if data collection is stopped
    if(data collection is stopped)
       stopped = TRUE;
    else if (counter == MAX_TIME_STOP)
       time_out = TRUE
  }
  if(timed out)
    kill dcm

 counter = 0;
 time out = FALSE;
 initiated CM/2
 While(not finished || timed out)
 {
    sleep 6 seconds
    counter ++
    test to see if CM/2 is finished
```

```
    if(CM/2 is finished)
       finished  = TRUE;
    else if (counter == MAX_FINISH_TIME)
       time_out = TRUE
   }
   if(timed out)
    return RC_ERROR

  counter = 0;
  time out = FALSE;
  initiated LAPS
  While(not finished || timed out)
  {
    sleep 6 seconds
    counter ++
    test to see if LAPS is finished
    if(LAPS is finished)
       finished  = TRUE;
    else if (counter == MAX_LAPS_FINISH_TIME)
       time_out = TRUE
   }
   if(timed out)
    return RC_ERROR

 return RC_OK
 }
```

1.2.8.4.7.1.2. Increment Phase

```
/*********************************************************************
*********
  FUNCTION:    IncrementPhase
  AUTHOR:      D. Hughes
  DESCRIPTION: Increments the current phase number in the upgrad.log file
  RETURNS:     The new phase number if successfully set
               -1 otherwise

 REVISIONS:
  DATE   NAME    DESCRIPTION
  ----------------------------------------------------------------
 08/21/97 D.Hughes    INITIAL REVISION

*********************************************************************
*********/
/*
IncrementPhase()
*/
LONG IncrementPhase( VOID )
{
  CHAR  szPhaseNumber[35];
  LONG  lPhase = 0;
```

```
  lPhase FindValue(log file path, PHASE, PHASE);  /* determine the phase
                                                     number */

  _ltoa(lPhase+1, szPhaseNumber,10);
  if(SearchAndAddStringtoFile (pszFileNamePath,
                                PHASE,PHASE,
                        szPhaseNumber) == RC_OK)
  {
    return ++lPhase;
  }
  return -1;
}
```

1.2.8.4.7.1.3. DiskSize

```
/*********************************************************************
*********
  FUNCTION:    DiskSize
  AUTHOR:      D. Hughes
  DESCRIPTION: Finds the size of the hard drive
  RETURNS:     Size of the hard drive

 REVISIONS:
  DATE   NAME    DESCRIPTION
  ----------------------------------------------------------------------
 08/08/97 D.Hughes    INITIAL REVISION

*********************************************************************
*********/
/*
DiskSize()
*/
LONG DiskSize( void )
{

  APIRET rc = NO_ERROR;

  FILE* pfDiskImageFile = NULL;

  HFILE hfDiskFIleHandle = 0; /* handle to C: drive                  */
  CHAR szFileName[20] = "C:";
  PCHAR pszFileName;
  ULONG ulAction;          /* paramters for DOSOPEN                 */
  ULONG ulCbFile = 0;      /*                                  */
  ULONG ulAttribute = 0;   /*                                  */
  ULONG ulFsOpenFlags = 0;  /*                                    */
  ULONG ulFsOpenMode = 0;   /*                                    */

  UCHAR uchLockParam = 0;   /* parameter for IOCTL lock             */
  UCHAR uchLockData = 0;    /* data for IOCTL lock                  */
  ULONG ulLockParamLenMax;  /* Max length of param sent IOCTL
*/
  ULONG ulLockPlength;      /* length of parameter sent IOCTL       */
```

```
PULONG  pulLockParmLen;/*  pointer length of param sent IOCTL      */
PVOID   pLockParams;

ULONG ulParamLenMax = 0; /*  Max length of param sent IOCTL         */
PULONG pulParmLen; /*  pointer length of param sent IOCTL      */
ULONG ulPLength = 0; /*  length of parameter sent IOCTL         */

ULONG ulDataLenMax = 0;     /*  Max length of data sent IOCTL          */
PULONG pulDataLen;          /*  pointer length of data sent IOCTL      */
ULONG ulDLength = 0;        /*  length of data sent IOCTL              */
APIRET ulrc;                /*  return code                            */
UCHAR uchParms[120];

struct Parameters  Params;   /* parameters structure for IOCTL          */
struct Parameters* pParams; /* pointer to parameters structure for IOCTL */
struct DeviceParams  DevParms;/* device structure for IOCTL             */
struct DeviceParams* pDeviceParams;/* pointer to device structure for IOCTL*/

pszFileName = szFileName;

pParams = malloc(sizeof(Params));
pDeviceParams = malloc(sizeof(DevParms));

pulDataLen = &ulDLength;

ulCbFile = 100L;
ulAttribute = 0L;
ulFsOpenFlags = 1L;
ulFsOpenMode =  OPEN_FLAGS_WRITE_THROUGH |
            OPEN_SHARE_DENYREADWRITE |
            OPEN_FLAGS_NOINHERIT |
            OPEN_FLAGS_DASD;

/* Get a handle for disk drive C: */
rc = DosOpen(pszFileName,
      &hfDiskFIleHandle,
      &ulAction,
      ulCbFile,
      ulAttribute,
      ulFsOpenFlags,
      ulFsOpenMode,
      NULL
            );

if (rc != NO_ERROR)
{
  exit(0);
}

/* lock everyone out from access to drive */

ulLockParamLenMax = sizeof(uchParms);
ulLockPlength = 0;
pulLockParmLen = &ulLockPlength;
```

```
pLockParams = &uchLockParam;
ulLockPlength = sizeof(uchLockParam);
ulDLength = sizeof(uchLockData);

ulrc = DosDevIOCtl(hfDiskFIleHandle,
                   IOCTL_DISK,
                   DSK_LOCKDRIVE,
                   &uchLockParam,
                   ulLockPlength,
                   &ulLockPlength,
                   &uchLockData,
                   ulDLength,
                   &ulDLength
                  );

 if (ulrc != NO_ERROR)
 {
   exit(0);
 }

ulPLength = sizeof(Params);
ulDLength = sizeof(sizeof(DevParms));
Params.ucCommand_info = 0x01;
Params.ucDrive_info = 0x02;
ulrc = DosDevIOCtl(hfDiskFIleHandle,
                   IOCTL_DISK,
                   DSK_GETDEVICEPARAMS,
                   pParams,
                   ulPLength,
                   &ulPLength,
                   pDeviceParams,
                   ulDLength,
                   &ulDLength
                  );
 if (ulrc != NO_ERROR)
 {
   exit(0);
 }

/* release lock on drive c: */

ulLockParamLenMax = sizeof(uchParms);
ulLockPlength = 0;
pulLockParmLen = &ulLockPlength;
pLockParams = &uchLockParam;
ulLockPlength = sizeof(uchLockParam);
ulDLength = sizeof(uchLockData);

ulrc = DosDevIOCtl(hfDiskFIleHandle,
                   IOCTL_DISK,
                   DSK_UNLOCKDRIVE,
                   &uchLockParam,
                   ulLockPlength,
                   &ulLockPlength,
```

```
                &uchLockData,
                ulDLength,
                &ulDLength
               );

  /* release the handle to drive c: */
  rc = DosClose(hfDiskFIleHandle);
  fclose(pfDiskImageFile);

return pDeviceParams->Num_Cylinders;

} /* end of DiskSize() */
```

1.2.8.4.7.1.4. PhaseZero

```
/*****************************************************************
******
*
* NAME:        PhaseZero
*
* DESCRIPTION:  Tests to see if the zip files are valid
*
* ASSUMPTIONS:
*
*---------------------------------------------------------------
 *
* REVISION HISTORY:
*
*   Date      Author      Description
*   ----------------------------------------------------------------
*   08/08/97   Doug Hughes   Original code
*****************************************************************
******/
PhaseZero()
{
  /*
          check to see if a valid os_drive.zip exists.  The upgrade.ini files
          in each zip file should be identical.  The test here is if the zip
          file is valid.  It is valid if it contains a upgrade.ini
    */
     if (os_drive.zip exists)
     {
       unzip -q  os_drive.zip upgrade.ini
       if( upgrade.ini exists)
       {
           /*
              Check to see if upgrade.ini is valid.  Compare ini version to
              upgrade.exe version
            */
            if ( iIniLevel != iExeLevel)
            {
               put error message to upgrade.log
               exit(0)
            }
```

```
        /*
          rename upgrade.ini to temp.ini for test on
          nextgen_drive.zip
        */
        rename upgrade.ini  to temp.ini
    }
    else
    {
       log error - invalid os_drive.zip
       exit
    }

  } /* end of if os_drive.zip exists */

  if (nextgen_drive.zip exists)
  {
    unzip  nextgen_drive.zip upgrade.ini
    if( upgrade.ini exists)
    {
        /*
            Check to see if upgrade.ini is valid.  Compare ini version to
            upgrade.exe version
        */
         if ( iIniLevel != iExeLevel)
         {
             put error message to upgrade.log
             exit(0)
         }

        /*
          rename upgrade.ini to temp.ini for test on
          boot_drive.zip
        */
        rename upgrade.ini  to temp.ini
    }
    else
    {
       log error - invalid nextgen_drive.zip
       exit
    }

  } /* end of if nextgen_drive.zip exists */

  if ( boot_drive.zip exists)
  {
    unzip boot_drive.zip upgrade.ini
    if( upgrade.ini exists)
    {
        /*
            Check to see if upgrade.ini is valid.  Compare ini version to
            upgrade.exe version
        */
         if ( iIniLevel != iExeLevel)
```

```
        {
            put error message to upgrade.log
            exit(0)
        }
    }
    else
    {
       log error - invalid boot_drive.zip
       exit
    }

  } /* end of if boot_drive.zip exists */

 if ( upgrade.ini does not exist && tmp.ini does}
 {
  /*
     Temp.ini exists and upgrade.ini does not exist
  */
    copy temp.ini upgrade.ini
 }

if (upgrade.ini  does not exist)
{
  Report error - no valid zip file
   exit
{
 else
 {
 read version number from upgrade.ini
  read version number from DCS 300
 compare the DCS 300 software version to the version controller in
      upgrade.ini
  if ( the DCS  300 software version is not in version controller)
  {
    log error message to upgrade.log
    quit upgrade
  }

  open upgrade.ini;
  read in upgrade type to pszUpgradeType;
  if( pszUpgradeType == Minor)
  {
    iUpgradeType = 0;
  }
  if(strcmp( pszUpgradeType,REBOOT))
  {
    iUpgradeType = 1;
  }
  if( strcmp(pszUpgradeType,SHUTDOWN))
  {
    iUpgradeType = 2;
  }
  if( strcmp(pszUpgradeType,SEVERE))
  {
```

```
 iUpgradeType = 3;
}

 if (iUpgradeType,SEVERE))
 {
   /*
      Find the size
      of the hard drive
      and log it.
   */
    lDiskSize = DiskSize();
    _ltoa(lDiskSize, szPhaseNumber,10);

    SearchAndAddStringtoFile (pszFileNamePath,
                              DISKSIZE, DISKSIZE,
                              lDiskSize)
   read in the additional parameters from upgradet.ini;
   make sure that the partition names are for the appropriate disk
   size using lDiskSize;
}
if(iUpgradeType == SHUTDOWN)
{
   read in the additional parameters from upgrade.ini
}
if (iUpgradeType == REBOOT || iUpgradeType == SEVERE)
/* get the list of system files to backup */
  if ( os_drive.zip exists)
  {
     extract UPGDBKUP from zip file
  }
  else if (nextgen.zip exists)
  {
    extract UPGDBKUP from nextgen.zip file
  }
  else if (boot.zip exists
  {
    extract UPGDBKUP from boot.zip file
  }
  else
  {
    log error - no valid zip file
    quit
  }
} /* end of get system file backup list */

/* no problems with *.zip and upgrade.ini.   */
/* also, no problems with version numbers. */
/* Proceed with the upgrade ....              */

 log success message to upgrade.log
 if (chParameter == 'q')
 {
   /* a query of the upgrade was made */
   exit
```

```
        }
        if (iUpgradeType == SEVERE)
        /* Skip to phase 1 */
        {
           phase = 1;
           write phase to log file
}
        else if (iUpgradeType == REBOOT)
        /* Skip to phase 4 */
        {
           phase = 4;
           write phase to log file
        }
        else
        {
           /* MINOR or SHUTDOWN upgrade */
           phase = 5;
           write phase to log file;
         }
     }
return;
}
```

1.2.8.4.7.1.5. PhaseOne

```
/*********************************************************************
******
*
* NAME:        PhaseOne
*
* DESCRIPTION:  Delete boot partition, Create boot partition
*                    delete/create boot manager
*
* ASSUMPTIONS:
*
*--------------------------------------------------------------------
 *
* REVISION HISTORY:
*
*   Date      Author      Description
*   ----------------------------------------------------------------
*   08/08/97   Doug Hughes   Original code
*
*********************************************************************
*/
PhaseOne()
{
    /*
        save the type of video
        driver to upgrad.ini
     */
    SaveVideoType()

     if(fSaveActivate)
```

```
    {
      check to see if save and activate is necessary
       if (save and activated needed)
       {
         lRC = SaveAndActivate();
          if (lRC != RC_OK)
          {
            Log error message to log file
          }

       }
    }

    check to see if mh que exists
    if(mh que exist)
    (
       /* shut down data collection if it is running */
       Shut down data collection.
       loop testing to see if mh que still exists
       if doesn't exist, quit looping.  If still exist,
       wait TBD time and then stop looping.
      If timed out, kill dcm.
   }
   if(OS == 1)
   {
     extract UPGDBKUP from zip file
     lRc = SystemBackup(d:\upgrade);

     if( lRc != RC_OK )
     {
        printf( SYSTEM_BACKUP_ERR);
        printf( "%ld", lRc );
        exit(0);
     } // end if

    lRc = CreateUpgradeTmpFile(d:\upgrade);
     if( lRc != RC_OK )
     {
       printf(UPGRAD_TMP_FILE_ERR );
       exit(0);
     }
   } /* end if OS == 1  */

   if(BOOT_MANAGER == 1)
   {
     create boot manager

   if(BOOT == 1)
   {

     /*
        boot partition changed sizes and swapper partition must be
        deleted
```

```
        */
        if(!strcmp(pszSwapperPartitionName,NULL))
        {
          lRc = DeletePartitions(pszSwapperPartitionName);
          if (lRC != NO_ERROR)
          {
            exit(0);
          } /* end if */
        }

        /*
          delete  the boot partition
        */
        if(!strcmp(pszBootPartitionName,NULL))
        {
          lRc = DeletePartitions(pszBootPartitionName);
          if (lRC != NO_ERROR)
          {
            exit(0);
          } /* end if */
        }

        if(lBootPartitionSize != 0)
        {
          lRc = Create_Partition(lBootPartitionSize,
                                 iBootPartitionType,
                                 iBootPartitionStartable,
                                 pszBootPartitionFS,
                                 chBootPartitionStart
                                );
          if (lRC != NO_ERROR)
          {
            exit(0);
          } /* end if */
        }
      } /* end if BOOT_PARTITION_CHANGED */

      CLEAR_SCREEN;
      printf(PHASE_MSG, lPhase, lPhase + 1);
      fflush(stdout);
      lPhase = IncrementPhase();

      if(BOOT  == 1)
      {
        Copy new config.sys and new startup.cmd;
       /* reboot to activate fdisk changes */
       reboot to c drive
      } /* end if BOOT_PARTITION_CHANGED */

    } /* end if phase 1 */

    Return;
  }
```

1.2.8.4.7.1.6. PhaseTwo

```
/*****************************************************************
********
*
* NAME:        PhaseTwo
*
* DESCRIPTION:  Format boot partition, delete/create  swapper
*                    OS and/or nextgen partitions copy boot   drive
*                    operating system files.
*
* ASSUMPTIONS:
*
*----------------------------------------------------------------
*
* REVISION HISTORY:
*
*   Date      Author      Description
*   ----------------------------------------------------------
*   08/08/97   Doug Hughes   Original code
*
*****************************************************************
******/
PhaseTwo()
{
      read in the additional parameters from upgradet.ini;
      make sure that the partition names are for the appropriate disk
      size using lDiskSize;
      if(BOOT == 1 && !strcmp( psz BootFileSystem,NULL))
      {

        FormatPartition(psz BootFileSystem);
        unzip bootpart.zip into the boot partition
        copy upgrade.log from d:\ugprade to f: drive
        copy over backup system files from d:\upgrade to f: drive
      }
      if(BOOT != 1
         && SWAPPER == 1 &&
         strcmp(pszSwapperPartitionName, NULL))
      {
        /*
           delete the swapper partition
        */
         lRc = DeletePartitions(szSwapperPartitionName);
         if (lRC != NO_ERROR)
         {
            exit(0);
         } /* end if */
      } /* end if  BOOT != 1SWAPPER == 1 &&
           !pszSwapperPartitionName= NULL */

      if(OS == 1 &&
           !strcmp(pszOSPartitionName, NULL) )
      {
```

```
    /*
       delete the os partition.
    */
    lRc = DeletePartitions(pszOSPartitionName);
    if (lRC != NO_ERROR)
    {
      exit(0);
    } /* end if */
} /* end if OS  == 1  && pszOSPartitionName != NULL */

if(NEXTGEN == 1 &&
     !strcmp(pszNextgenPartitionName, NULL) )
    {
    /*
       delete the nextgen partition
    */
    lRc = DeletePartitions(pszNextgenPartitionName);
    if (lRC != NO_ERROR)
    {
      exit(0);
    } /* end if */
} /* end if NEXTGEN == 1 && pszNextgenPartitionName!= NULL */

if(SWAPPER == 1 &&
     lSwapperPartitionSize != 0 )
{
  lRc = Create_Partition(lSwapperPartitionSize,
                         iSwapperPartitionType,
                         iSwapperPartitionStartable,
                         pszSwapperPartitionFS,
                         chSwapperPartitionStart
                        );
  if (lRC != NO_ERROR)
  {
    exit(0);
  } /* end if */
} /* end if SWAPPER == 1 && lSwapperPartitionSize != 0  */

if(OS  == 1 &&
     lOSPartitionSize != 0)
{
  lRc = Create_Partition(lOSPartitionSize,
                         iOSPartitionType,
                         iOSPartitionStartable,
                         pszOSPartitionFS,
                         chOSPartitionStart
                        );
  if (lRC != NO_ERROR)
  {
    exit(0);
  } /* end if */

} /* end if OS == 1  && lOSPartitionSize != 0*/
```

```
    if(NEXTGEN  == 1 &&
            lNextgenPartitionSize != 0
     )
    {
      lRc = Create_Partition(lNextgenPartitionSize,
                             iNextgenPartitionType,
                             iNextgenPartitionStartable,
                             pszNextgenrPartitionFS,
                             chNextgenPartitionStart
                            );
      if (lRC != NO_ERROR)
      {
         exit(0);
      } /* end if */

    } /* end if NEXTGEN  == 1 && lOSPartitionSize != 0 */

    increment phase in upgrade.log on f: drive  to phase 3

    if(NEXTGEN  == 1 ||
              OS == 1 ||
          SWAPPER == 1)
    {
      fdisk /IBD:BOOT_DRIVE;
    }


  Return;
}
```

1.2.8.4.7.1.7. PhaseThree

```
/*****************************************************************
******
*
* NAME:        PhaseThree
*
* DESCRIPTION:  Format boot partition, delete/create  swapper
*                    OS and/or nextgen partitions
*
* ASSUMPTIONS:
*
*---------------------------------------------------------------
*
* REVISION HISTORY:
*
*   Date      Author       Description
*   ----------------------------------------------------------
*   08/08/97   Doug Hughes   Original code
*
*****************************************************************
****/
PhaseThree()
{
```

```
  if (!strcmp(pszOSPartitionFS, NULL)
  {
    FormatPartition(C_DRIVE,pszOSPartitionFS);
  } /* end if pszOSPartition != NULL */

  if (!strcmp(pszNextgenPartitionFS, NULL)
  {
    FormatPartition(D_DRIVE,pszNextgenPartitionFS);
  } /* end if pszNextgenPartition != NULL */

  if (!strcmp(pszSwapperPartitionFS, NULL)
  {
    FormatPartition(D_DRIVE,pszSwapperPartitionFS);
  } /* end if pszSwapperPartition != NULL */

  increment phase in upgrade.log  to phase 5
 Return;
}
```

1.2.8.4.7.1.8. PhaseFour

```
/*****************************************************************
******
*
* NAME:        PhaseFour
*
* DESCRIPTION:  PhaseFour is for REBOOT upgrades.  It does
*               A save and activate and reboot to the alternate
*               operating system
* ASSUMPTIONS:
*
*-----------------------------------------------------------------
 *
* REVISION HISTORY:
*
*   Date      Author      Description
*   ----------------------------------------------------------------
*   08/08/97  Doug Hughes  Original code
*
*****************************************************************
*******/
PhaseFour()
{
  if(fSaveActivate)
   {
      check to see if save and activate is necessary
      if (save and activated needed)
      {
        lRC = SavaAndActivate();
        if (lRC != RC_OK)
        {
          Log error message to log file
        }
      }
```

```
    }

    check to see if mh que exists
    if(mh que exist)
    (
      /* shut down data collection if it is running */
      Shut down data collection.
      loop testing to see if mh que still exists
      if doesn't exist, quit looping.  If still exist,
      wait TBD time and then stop looping.
      If timed out, kill dcm.
    }

  fdisk /IBD:BOOT_DRIVE

  return
}
```

1.2.8.4.7.1.9. PhaseFive

```
/*****************************************************************
************
*
* NAME:        PhaseFive
*
* DESCRIPTION:  For reboot upgrades, a system backup is done.
*                For all upgrades, the zip files are unzipped.
*                For the reboot and severe upgrades,  the system
*                configuration is restored
* ASSUMPTIONS:
*
*---------------------------------------------------------------
*
* REVISION HISTORY:
*
*   Date      Author      Description
*   ----------------------------------------------------------
*   08/08/97   Doug Hughes   Original code
*****************************************************************
*********/
PhaseFive()
{

    if (iUpgradeType == SHUTDOWN)
    /* this is an upgrade primarily of the nextgen directory */
    {
     //  Shutdown nextgen processes (dcm stop?)
    ; /*
       This will not be implemented until the DCS 300
       migrates to Windows NT
      */
    }

    if (iUpgradeType == REBOOT)
```

```
/*
   Usually a Major upgrade of c: drive - need to back up system
  files, run creat_ng
 */
{
   SaveVideoType()

   extract UPGDBKUP from zip file
   lRc = SystemBackup(d:\upgrade);

   if( lRc != RC_OK )
   {
      printf( SYSTEM_BACKUP_ERR);
      printf( "%ld", lRc );
      exit(0);
   } // end if

  lRc = CreateUpgradeTmpFile(d:\upgrade);
   if( lRc != RC_OK )
   {
     printf(UPGRAD_TMP_FILE_ERR );
     exit(0);
   }
} /* end of if REBOOT */

CLEAR_SCREEN;
printf(WORKING_MSG);
fflush(stdout);

//
//  Copy D: drive from CD ROM
//

//   Make sure we are on the D drive
//
lRc = DosSetDefaultDisk( 4 ); // Set to the D drive (1=A,2=B,...)
if( lRc != RC_OK )
{
  printf(SETTING_DRIVE_ERR);
  printf("%d", lRc);
  exit(0);
}
// Move to the root directory
lRc = DosSetCurrentDir("\\");

printf(COPY_SYSTEM_FILES_INFO);

lRc = DosExecPgmWrapper(UNZIP_EXE, D_IMAGE);
if (lRC != NO_ERROR)
{
  printf(UNZIPING_FILES_TO_D_ERR);
  exit(0);
} // end if
```

```
//   Make sure we are on the C drive
//
lRc = DosSetDefaultDisk( 3 ); // Set to the C drive (1=A,2=B,...)
if( lRc != RC_OK )
{
  printf(SETTING_DRIVE_ERR);
  printf("%d", lRc);
  exit(0);
}
// Move to the root directory
lRc = DosSetCurrentDir("\\");

lRc = DosExecPgmWrapper(UNZIP_EXE, C_IMAGE);
if (lRC != NO_ERROR)
{
  printf(UNZIPING_FILES_TO_C_ERR);
  exit(0);
} // end if

 if iUpgradeType == SHUTDOWN)
/* this is an upgrade primarily of the nextgen directory */
{
   TBD
   Restart nextgen processes (dcm start?)
}


if (iUpgradeType == REBOOT || iUpgradeType == SEVERE )
{
  // ************************************************
  // Before running creat_ng, copy upgrade temp file
  // from d:\ugprade drive to location where creat_ng looks for
  // it.
  // ************************************************
  memset(szSysCmdStr,NUL,
           NG_BUF_IN_LEN + NG_BUF_IN_LEN +2);

  sprintf(szSysCmdStr,"%s %s%c%s%c%s %s%c%s%s%s",
  COPY_COMMAND,
                    E_DRIVE,
                    OSCHAR_DIR,
                    UPGRADE_DIR,     // Where we saved it
                    OSCHAR_DIR,
                    DEFAULT_NG_TEMP_UPGRAD_FILE,
                     KNOWN_NG_SYS_INI_PATH,  // where it needs
                                                    to be
                    OSCHAR_DIR,
                    DEFAULT_NG_TEMP_UPGRAD_FILE,
                    REDIRECT_OUTPUT,NUL_FILE);  // keep it quiet

  iSysReturn = system(szSysCmdStr);

  if (iSysReturn < 0)
  {
```

```
    /* try a DosCopy, see if that works -
     * it will give us better error return code
     */
    sprintf(szSysCmdStr,"%s%c%s%c%s",  E_DRIVE,
                        OSCHAR_DIR,
                        UPGRADE_DIR,     // Where we saved it
                        OSCHAR_DIR,
                        DEFAULT_NG_TEMP_UPGRAD_FILE);

    // where it needs to be
    sprintf(szFileNamePath, "%s%c%s", KNOWN_NG_SYS_INI_PATH,
                         OSCHAR_DIR,
                         DEFAULT_NG_TEMP_UPGRAD_FILE);

    iSysReturn = DosCopy(szSysCmdStr, szFileNamePath, 0L);
    if(iSysReturn != 0)
    {
      sprintf(szErrorOut,"%s\n%s\n%s %s\nError Code:%d\n%s",
                              ERROR_HEADING_300_D,
                              COPY_ERR,
                              szSysCmdStr, szFileNamePath,
                              iSysReturn,
                              CONTACT_INTER_SUPPORT);

      printf("\n\n%s\n\n",szErrorOut);
      exit(0);
    }
  } // end if

//   Make sure we are on the d:: drive
//
lRc = DosSetDefaultDisk(4); // Set to the f: drive (1=A,2=B,...)
if( lRc != RC_OK )
{
  printf(SETTING_DRIVE_ERR);
  printf("%d", lRc);
  exit(0);
}
// now call creat_ng to update the configuration
lRc = DosExecPgmWrapper(CREAT_NG_EXE, "");

if( lRc != RC_OK )
 {// If failure here, we expect creat_ng would have listed
      errors on its own.

  printf(  "\ntermcodpid = %ld, resultcode = %ld",
        ChildRC.codeTerminate, ChildRC.codeResult );

       exit(0);
 }

// **********************************
// restore backed up system files
// **********************************
```

```
    lRc = SystemRestore(d:\upgrade);

    CLEAR_SCREEN;
    printf(WORKING_MSG);
    fflush(stdout);

    // ************************************************
    // Restore old TCPIP start command to config.sys file.
    // ************************************************
    if (EnableTcpipInConfigSys() != RC_OK)
    {
       exit(0);
    }
    // ****************************************
    // Restore old HOSTNAME to config.sys file.
    // ****************************************
    HostNameRestore(f:\upgrade);  // Don't check return code,
                                  // always returns OK

    /* copies the correct drivers for the video card */
    CopyVideoDrivers();

    /*
        puts the correct lines in config.sys and
        config.ngc for the video card
     */
    FixVideoInConfigSys();

  } /* end if REBOOT || SEVERE */

  if ( REBOOT || SEVERE)
  {
    fdisk /startabl /name:c
     /* force reboot  to c: drive*/
    fdisk /ibd:c;
   }

  return
}
```

1.2.8.4.7.1.10. SaveVideoType

```
/*********************************************************************
************
* NAME: SaveVideoType
*
* DESCRIPTION:  Finds the type of video drivers in config.sys and
*                    stores the value in upgrade.log
* ASSUMPTIONS:
*
*-------------------------------------------------------------------
*
* REVISION HISTORY:
*
```

```
*   Date      Author      Description
*   ----------------------------------------------------------------
*   08/08/97   Doug Hughes   Original code
*********************************************************************
*/
SaveVideoType()
{
      detect type of video card
      search config.sys for VIDEO_DEVICE
      on same line look for VIO_SVGA or VIO_VGA
      Store the result in upgrade.log
      use old find value.
     result
}
```

1.2.8.4.7.1.11. CopyVideoDrivers

```
/*********************************************************************
************
* NAME: CopyVideoDrivers
*
* DESCRIPTION:  Copies the Video Drivers to the c:\os2\dll and
*                    C:\os2\mdos directories
* ASSUMPTIONS:   booted from the alternate boot drive
*
*-------------------------------------------------------------------
*
* REVISION HISTORY:
*
*   Date      Author      Description
*   ----------------------------------------------------------
*   08/08/97   Doug Hughes   Original code
*********************************************************************
*****/
CopyVideoDrivers()
{
   Read video type from upgrade.log
   if( videotype == SVGA)
   {
      copy  these files from the SVGA directory to the c:\os2\dll directory:
         BVHSVGA.DLL
         BVHVGA.DLL
         IBMDEV32.DLL
         IBMVGA32.DLL
         copy VSVGA.SYS from the SVGA directory to the c:\os2\mdos
         directory.
   }
   if( videotype == VGA)
   {
      copy  these files from the VGA directory to the c:\os2\dll directory:
         BVHSVGA.DLL
         BVHVGA.DLL
         IBMDEV32.DLL
         IBMVGA32.DLL
```

```
            copy VSVGA.SYS from the SVGA directory to the c:\os2\mdos
            directory.
        }

    }
    FixVideoInConfigSys()
    /*********************************************************************
    ************
    * NAME: FixVideoInConfigSys()
    *
    * DESCRIPTION:   Fixes the settings in config.sys for the video
    *
    * ASSUMPTIONS:   none
    *
    *--------------------------------------------------------------------
     *
    * REVISION HISTORY:
    *
    *   Date      Author      Description
    *   ----------------------------------------------------------------
    *   08/08/97   Doug Hughes   Original code
    *********************************************************************
    *****/
    FixVideoInConfigSys()
    {
        Read video type from upgrade.log
        if( videotype == SVGA)
        {
           copy SVGA lines to config.sys and config.ngc
        }

        if( videotype == VGA)
        {
           copy VGA lines to config.sys and config.ngc
        }

    }
```

1.2.8.4.7.1.12. DeletePartitions

```
    /*********************************************************************
    ***
    * NAME:        DeletePartitions
    *
    * DESCRIPTION:   deletes c and d partitions
    *
    * ASSUMPTIONS:
    *
    *--------------------------------------------------------------------
     *
    * REVISION HISTORY:
    *
    *   Date      Author      Description
    *   ----------------------------------------------------------------
```

```
*    08/08/97   Doug Hughes   Original code
*************************************************************
*******/
```

```
/*
DeletePartitions()
*/
LONG DeletePartitions (
                PCHAR pszDiskName
)
{
  LONG lRC = 0;
  CHAR  szSysCmdStr[NG_BUF_IN_LEN + NG_BUF_IN_LEN +2];  /* for
system    */
                                        /* copy command   */
  CHAR  szErrorOut[MAX_ERROR_OUT_LEN + 1]; /* for error output strings
*/
  INT   iSysReturn = 0;                 /* for return from system call */

  /*
      The Partition names are different for the 540Meg, 2.2 Gig,
       and 2.5 Gig drives
  */

    memset(szSysCmdStr,NUL,NG_BUF_IN_LEN + NG_BUF_IN_LEN +2);

    sprintf(szSysCmdStr,"%s%s%s%s%s",FDISK_COMMAND,
DELETE_COMMAND,
                        pszDiskName,
                        REDIRECT_OUTPUT, NUL_FILE);

    iSysReturn = system(szSysCmdStr);

    if (iSysReturn < 0)
    {
      lRC = -1;


sprintf(szErrorOut,"%s\n%s\n%s\n%s",ERROR_HEADING_300_D,DELETE_DR
IVE_ERR,
                              szSysCmdStr,
                              CONTACT_INTER_SUPPORT);

      printf("\n\n%s\n\n",szErrorOut);
      fflush(stdout);

    } /* end if */

  return lRC;

} /* end of DeletePartitions() */
```

1.2.8.4.7.1.13. FormatPartitions

```
/*********************************************************************
****
* NAME:        FormatPartition
*
* DESCRIPTION:  format passed in drive name
*
* ASSUMPTIONS:
*
*--------------------------------------------------------------------
*
* REVISION HISTORY:
*
*   Date      Author      Description
*   ----------------------------------------------------------------
*   08/08/97   Doug Hughes   Original code
*   08/22/97 D. Kaatz/D.H.  Made more generic; pass in drive to part.
*********************************************************************
***********/

/*
FormatPartition()
*/
LONG FormatPartition (PCHAR pszDrive, PCHAR pszFileSystem)
{
  LONG lRC = 0;
  CHAR  szSysCmdStr[NG_BUF_IN_LEN + NG_BUF_IN_LEN +2];  /* for
system      */
                                        /* command commands  */
  CHAR  szErrorOut[MAX_ERROR_OUT_LEN + 1]; /* for error output strings
*/
  INT   iSysReturn = 0;                 /* for return from system call    */


  /* format drive */
  memset(szSysCmdStr,NUL,NG_BUF_IN_LEN + NG_BUF_IN_LEN +2);
  sprintf(szSysCmdStr,"%s%s %s%s%s%s%s",
                        FORMAT_COMMAND, pszDrive,
                        pszFileSystem,
                        REDIRECT_INPUT,
                        INPUT_FILE,
                        REDIRECT_OUTPUT,
                        NUL_FILE);

  iSysReturn = system(szSysCmdStr);


  if (iSysReturn < 0)
  {
    lRC = -1;
    sprintf(szErrorOut,"%s\n%s\n%s\n%s",
                        ERROR_HEADING_300_D,
```

```
                        FORMAT_DRIVE_ERR,
                        szSysCmdStr,
                        CONTACT_INTER_SUPPORT);

    printf("\n\n%s\n\n",szErrorOut);
    fflush(stdout);

  } /* end if */

  return lRC;
} /* end FormatPartitions() */
```

1.2.8.4.7.1.14. CreatePartitions

```
/*********************************************************************
******
* NAME:        CreatePartitions
*
* DESCRIPTION:  Creates new partition
*
* ASSUMPTIONS:
*
*-------------------------------------------------------------
*
* REVISION HISTORY:
*
*   Date      Author      Description
*   ----------------------------------------------------------
*   11/12/97  Doug Hughes  Original code
*
*********************************************************************
***/

/*
CreatePartitions()
*/
LONG CreatePartitions(
              LONG lDiskSize,
              LONG lType,
              INT iStartable,
              PSZCHAR pszFileSystem,
              CHAR cStart
)
{
  LONG lRC = 0;
  CHAR  szSysCmdStr[NG_BUF_IN_LEN + NG_BUF_IN_LEN +2];  /* for
                                          system copy command  */
  CHAR  szErrorOut[MAX_ERROR_OUT_LEN + 1]; /* for error output
                                              strings   */
  INT   iSysReturn = 0;            /* for return from system call */

 sprintf(szSysCmdStr,"%s%s%s%s%s%s%s%s%s%s%s%s%s",
                                        FDISK_COMMAND,
                                      CREATE_COMMAND,
```

```
                                        FSTYPE, pszFileSystem,
                                              VTYPE, lType,
                                        DISK_SIZE, lDiskSize,
                                              START, cStar,
                                        REDIRECT_OUTPUT,
                                                    NUL_FILE);

    iSysReturn = system(szSysCmdStr);

    if (iSysReturn < 0)
    {
      lRC = -1;

      sprintf(szErrorOut,"%s\n%s\n%s\n%s",ERROR_HEADING_300_D,
                                          CREATE_DRIVE_ERR,
                                        szSysCmdStr,
                                      CONTACT_INTER_SUPPORT);

      printf("\n\n%s\n\n",szErrorOut);
      fflush(stdout);

    } /* end if */

  return lRC;
  } /* end of CreatePartitions() */
```

1.2.8.4.7.1.15. FindValue

```
  /*********************************************************************
  *********
  * NAME:        FindValue
  *
  * DESCRIPTION:  find value of input string in a file under
  *                    specified section
  *
  * ASSUMPTIONS:  none
  *
  *-------------------------------------------------------------------------
  *
  * REVISION HISTORY:
  *
  *   Date      Author      Description
  *   ----------------------------------------------------------------
  *   08/08/97   Doug Hughes   Original code
  *********************************************************************
  ********/

  /*
  FindValue()
  */
  LONG FindValue (
          PSZ  pszFilePath,
```

```
        PSZ  pszSEction,
        PCHAR szFindString
)
{
  FILE  *pfFile   = NULL;          /* file ptr for file     */
/* env var                 */
  CHAR   szCurLine[NG_BUF_IN_LEN + 1];   /* buffer for file reads       */
  CHAR   szLine[NG_BUF_IN_LEN +1];       /* used for processing file line */
  PCHAR  pszTemp = NULL;
  BOOL fSection = FALSE;
  BOOL fSetting = FALSE;

  ULONG  ulLen   = 0;              /* buf length for env var       */
  LONG   lRC = RC_OK;                /* return code                */


  PCHAR  pszFileName = NULL;             /* ptr original file         */


  if (pszFileName != NULL)
  {

    pfFile = fopen(pszFilePath,FO_READ_ONLY);

    if ( pfFile != NULL)
    {
      memset(szCurLine,NUL,NG_BUF_IN_LEN + 1);

      while ( (fgets(szCurLine,NG_BUF_IN_LEN,pfFile) != NULL) )
      {
        if(fSection != TRUE)
        {
          if (szCurline[0] == '['
          {
            szCurline = szCurline + sizeof(CHAR);
            if(strncmp(szCurline,pszSection, strlen(pszSection)))
            {
              fSection = TRUE;
            }
          } /* end if szCurline[0] == '[' */
        } /* end if fSection != TRUE */
        else
        {
          if(strstr(szCurline, pszSetting) != NULL)
          {
            fSetting = TRUE;
            pszTemp = strstr(szCurline, '=');
            while(strcmp(pszTemp, ' ');
            {
              pszTemp = pszTemp + sizeof(CHAR);
            }
            pszValue = strcpy(psTemp);
          } /* end if */
        } /* end else */
```

```
        } /* end while */
      } /* end if pfFile != NULL */
  } /*  pszFileName != NULL */

  if (pfFile != NULL)
  {
    fclose(pfFile);
  }

  return( pszValue);

} /* end of FindValue() */
```

1.2.8.4.7.1.16. SearchAndAddStringtoFile

```
/*****************************************************************
********
* NAME:        SearchAndAddStringtoFile
*
* DESCRIPTION:  Find string in file and update it with the value
*               passed in.  If the sting is not found add it at
*               the end of the file.
*
* ASSUMPTIONS:  The file exists
*
*----------------------------------------------------------------------------
*
* REVISION HISTORY:
*
*   Date      Author      Description
*   ----------------------------------------------------------------------
*   08/08/97  Doug Hughes  Original code
*****************************************************************
******/

/*
SearchAndAddStringtoFile()
*/
LONG SearchAndAddStringtoFile (
          PSZ pszFileNamePath,
          PSZ    pszSection,
          PSZ pszSearchString,
          PSZ pszInsertString
)
{
  FILE  *pfLogFile   = NULL;        /* file ptr for  file        */
  FILE  *pfFileTemp = NULL;          /* file ptr for Original file  */

  PCHAR  pszTempPath = NULL;          /* temp file buffer          */

  CHAR  szTempPath[MAXPATH +1];       /* ptr to temp file          */
/* env var                */
```

```
CHAR   szCurLine[NG_BUF_IN_LEN + 1];    /* buffer for file reads        */
CHAR   szTempLine[NG_BUF_IN_LEN +1];    /* used for processing file
                                           line */

PCHAR  pszTemp = NULL;                  /* used for file writes         */
PCHAR  pszTempStart = NULL;             /* points to start of temp buffer*/
INT    iStartIndex = 0;                 /* starting point on line where  */
                                        /* replacement value should be   */
                                        /* placed                        */

ULONG  ulLen   = 0;                     /* buf length for env var       */
LONG   lRC = RC_OK;                     /* return code                  */
BOOL   fFoundString = FALSE;
INT    iLocalInstance = 1;
BOOL fSection = FALSE;
BOOL fSetting = FALSE;


/*****************************************
* Build full paths for both the original *
* file and the new file                  *
*****************************************/

if (pszFileNamePath != NULL)
{

  remove(LOG_TEMP_NAME);
  rename(pszFileNamePath, LOG_TEMP_NAME);

  pfLogFile   = fopen(pszFileNamePath, FO_WRITE);
  pfFileTemp = fopen(LOG_TEMP_NAME, FO_READ_ONLY);
  if ( pfLogFile != NULL )
  {

    /***********************************************
    * Now both files are open do the read and write *
    ***********************************************/

    memset(szCurLine,NUL,NG_BUF_IN_LEN + 1);
    memset(szTempLine,NUL,NG_BUF_IN_LEN + 1);
    pszTempStart = szTempLine;

    while ( (fgets(szCurLine,NG_BUF_IN_LEN,pfFileTemp) != NULL) )
    {

      memcpy(szTempLine, szCurLine, NG_BUF_IN_LEN + 1);
      if(fSection != TRUE)
      {
        if (szCurline[0] == '['
        {
          pszTemp = szCurline + sizeof(CHAR);
          if(strncmp(pszTemp,pszSection, strlen(pszSection)))
          {
            fSection = TRUE;
```

```
            }
          } /* end if szCurline[0] == '[' */
          /*  write the lines to the new file */
          fprintf(pfLogFile, pszCurLine);
        } /* end if fSection != TRUE */
        else
        {
          if( fSetting != TRUE)
          {
            if(strstr(szCurline, pszSetting) != NULL)
            {
              /*
                 found the setting under the section
                 write the new value to the file
               */
              fSetting = TRUE;
              pszTemp = strstr(szCurline, '=');
              pszTemp = pszTemp + sizeof(CHAR);
              *pszTemp = NULL;
              strcat(pszTemp,' ');
              pszTemp = pszTemp + sizeof(CHAR);
              sprintf(pszTemp,"%s%s %s \n",pszSearchString,
                                                  ' = ',  pszInsertString);
              fprintf(pfLogFile, pszTemp);
              fFoundString = TRUE;
            }  /* end if */
          } /* end if fSetting != TRUE */
          else
          {
            /*
                string under the section  already found
                so copy the rest of the file to the new file
             */
            fprintf(pfLogFile, pszCurLine);
          }
        } /* end else */


      } /* end while */
    } /* end if pfFile != NULL */
} /*  pszFileName != NULL */


} /* end if */
else
{
  CHAR  szErrorOut[MAX_ERROR_OUT_LEN + 1];/* for error output
                                                    strings*/

  lRC = RC_FAIL_OPEN;
  sprintf(szErrorOut,"%s\nError opening log file.n%s",
                        ERROR_HEADING_300_D,
                        CONTACT_INTER_SUPPORT);
```

```
        printf("\n\n%s\n\n",szErrorOut);
        fflush(stdout);
        return lRC;

      } /* end else */

    } /* end if */


    if (fFoundString == FALSE)
    {
      lRC = -1
    }

    if (pfLogFile != NULL)
    {
      fclose(pfLogFile );
    }

    if (pfFileTemp != NULL)
    {
      fclose(pfFileTemp);
    }

    return(lRC);

  } /* end of SearchAndAddStringtoFile() */
```

1.2.8.4.7.1.17. EnableTcpipInConfigSys

```
  /*****************************************************************
  ***********
   NAME:       EnableTcpipInConfigSys

   AUTHOR:     David Kaatz

  DESCRIPTION: Look in the backup version of config.sys (config.bac),
  which  was created by creat_ng, for MPTSTART.  If MPTSTART
  was found in config.bac, update config.sys with MPTSTART
  command.  If no MPTSTART line is found, not an error, just return
  RC_OK.

   ASSUMPTIONS:  config.bac exists.

   REVISIONS:
    DATE    NAME    DESCRIPTION
    ----------------------------------------------------------------
   07/09/96  D.KAATZ     INITIAL REVISION

  *****************************************************************
  *********/
  LONG EnableTcpipInConfigSys(VOID)
```

```
{
  CHAR szNewConfSysFile[MAXPATH + 1];
  CHAR szBkConfSysFile[MAXPATH + 1];
  CHAR szConfSysFile[MAXPATH + 1];
  CHAR szCurLine[MAX_CONFIG_LINE_LEN + 1];
  CHAR szErrorOut[MAX_CONFIG_LINE_LEN + 1];
  PCHAR pszTemp;
  FILE *pfConfig  = NULL;
  FILE *pfConTarg  = NULL;
  LONG lRc        = RC_OK;
  BOOL fLineFound  = FALSE;

  memset(szNewConfSysFile, NUL, MAXPATH + 1);
  memset(szBkConfSysFile,  NUL, MAXPATH + 1);
  memset(szConfSysFile,    NUL, MAXPATH + 1);
  memset(szCurLine,  NUL, MAX_CONFIG_LINE_LEN + 1);

  sprintf(szConfSysFile, "%s%s", KNOWN_NG_CONFIG_SYS_PATH,
                        DEFAULT_CONFIG_SYS_FILE);

  sprintf(szBkConfSysFile, "%s%s", KNOWN_NG_CONFIG_SYS_PATH,
                        BACK_CONFIG_SYS_FILE);

  sprintf(szNewConfSysFile, "%s%s", KNOWN_NG_CONFIG_SYS_PATH,
                        TEMP_CONFIG_SYS_FILE);

  // ********************************************
  // Open the backup config file
  // ********************************************
  pfConfig  = fopen(szBkConfSysFile, FO_READ_ONLY);

  // ********************************************
  // Read it until we find the "MPTSTART" line.
  // Watch out for remarks lines.
  // ********************************************
  while( (fgets(szCurLine, MAX_CONFIG_LINE_LEN, pfConfig) != NULL)
       &&
       !fLineFound )
  {
    if( (pszTemp = strstr(szCurLine, MPTSTART)) != NULL &&
       strncmp(szCurLine, "rem", 3) && strncmp(szCurLine, "REM", 3) )
    {
      fLineFound = TRUE;
    }
  }
  fclose(pfConfig);
  pfConfig = NULL;
  if( !fLineFound )
  {
    return RC_OK;
  }

  fLineFound = FALSE;
```

```
  // ********************************************
  // Open the config.sys file
  // ********************************************

  pfConfig = fopen(szConfSysFile, FO_APPEND);

  if (pfConfig != NULL)
  {

    // ********************************************
    // Copy the new command into the config.sys
    // ********************************************

    fputs(MPT_START_COMMAND, pfConfig);
    fputs("\n\n",pfConfig);
    fclose(pfConfig);
  } // end if
  else
  {
    lRc = -1;
    sprintf(szErrorOut,"%s\n%s\n%s\n%s",ERROR_HEADING_300_D,
                          OPEN_ERR,
                          szConfSysFile,
                          CONTACT_INTER_SUPPORT);
    printf("\n\n%s\n\n",szErrorOut);
    fflush(stdout);
  }

  return lRc;

} /* end of EnableTcpipInConfigSys */
```

1.2.8.4.7.1.18. HostNameRestore

```
/*****************************************************************
***********
 NAME:       HostNameRestore

 AUTHOR:     David Kaatz

 DESCRIPTION: Look in the backup version of config.sys (config.bac),
         which  was created by creat_ng, for the HOSTNAME= line.
          it  to a new version of config.sys, replacing the default
          HOSTNAME= line that is there.  If no HOSTNAME line is
          found, not an error, just return RC_OK.

 ASSUMPTIONS:  config.bac exists.

 REVISIONS:
  DATE    NAME    DESCRIPTION
  ----------------------------------------------------------------
 07/09/96 D.KAATZ     INITIAL REVISION
```

```
*****************************************************************
*********/
LONG HostNameRestore(void)
{
  CHAR szNewConfSysFile[MAXPATH + 1];
  CHAR szBkConfSysFile[MAXPATH + 1];
  CHAR szConfSysFile[MAXPATH + 1];
  CHAR szCurLine[MAX_CONFIG_LINE_LEN + 1];
  CHAR szHostName[MAX_CONFIG_LINE_LEN + 1];
  PCHAR pszTemp;
  FILE *pfConfig  = NULL;
  FILE *pfConTarg  = NULL;
  LONG lRc        = RC_OK;
  BOOL fLineFound  = FALSE;

  memset(szNewConfSysFile, NUL, MAXPATH + 1);
  memset(szBkConfSysFile,  NUL, MAXPATH + 1);
  memset(szConfSysFile,    NUL, MAXPATH + 1);
  memset(szCurLine,  NUL, MAX_CONFIG_LINE_LEN + 1);
  memset(szHostName, NUL, MAX_CONFIG_LINE_LEN + 1);

  sprintf(szConfSysFile, "%s%s", KNOWN_NG_CONFIG_SYS_PATH,
                        DEFAULT_CONFIG_SYS_FILE);

  sprintf(szBkConfSysFile, "%s%s", KNOWN_NG_CONFIG_SYS_PATH,
                        BACK_CONFIG_SYS_FILE);

  sprintf(szNewConfSysFile, "%s%s", KNOWN_NG_CONFIG_SYS_PATH,
                        TEMP_CONFIG_SYS_FILE);

  // ********************************************
  // Open the backup config file
  // ********************************************
  pfConfig  = fopen(szBkConfSysFile, FO_READ_ONLY);

  // ********************************************
  // Read it until we find the HOSTNAME= line.
  // Watch out for remarks lines.
  // ********************************************
  while( (fgets(szCurLine, MAX_CONFIG_LINE_LEN, pfConfig) != NULL)
       &&
       !fLineFound )
  {
    if( (pszTemp = strstr(szCurLine, NG_TCP_HOST_NAME)) != NULL &&
       strncmp(szCurLine, "rem", 3) && strncmp(szCurLine, "REM", 3) )
    {
      fLineFound = TRUE;
      strcpy( szHostName, szCurLine );
    }
  }
  fclose(pfConfig);
  pfConfig = NULL;
  if( !fLineFound )
  {
```

```
  return RC_OK;
}

fLineFound = FALSE;

// ***********************************************
// Open the config.sys file
// ***********************************************
pfConfig = fopen(szConfSysFile, FO_READ_ONLY);

// ***********************************************
// Open the temporary file
// ***********************************************
pfConTarg = fopen(szNewConfSysFile, FO_WRITE);

// ***********************************************
// Copy the config.sys into the temporary file
// until the HOSTNAME= line is reached, then
// insert the previously found hostname, then
// copy the rest of the file into the temp file.
// ***********************************************
// ***********************************************
// Read and copy except for the HOSTNAME= line.
// Watch out for remarks lines.
// ***********************************************
pszTemp = fgets(szCurLine, MAX_CONFIG_LINE_LEN, pfConfig);
while( pszTemp != NULL )
{
  if( strstr(szCurLine, NG_TCP_HOST_NAME) != NULL &&
      strncmp(szCurLine, "REM", 3) &&
      strncmp(szCurLine, "rem", 3) )
  {
    fputs(szHostName, pfConTarg);
  }
  else
  {
    fputs(szCurLine, pfConTarg);
  }
  pszTemp = fgets(szCurLine, MAX_CONFIG_LINE_LEN, pfConfig);
}
fclose(pfConfig);
fclose(pfConTarg);


// ***********************************************
// Delete config.sys, and rename the temp file
// to config.sys.
// ***********************************************
remove(szConfSysFile);
lRc = (LONG) rename(szNewConfSysFile, szConfSysFile);

return lRc;
} /* end of HostNameRestore */
```

1.2.8.4.7.1.19. SystemBackup

```
/*****************************************************************
***********
 NAME:       SystemBackup

 AUTHOR:     David Kaatz

 DESCRIPTION: Backup predefined list of files to d:\upgrade\backup.
          No error if file does not exist.
          Error if d:\upgrade\backup is not writable.

 ASSUMPTIONS: We always backup to d:\upgrade\backup.
          No blank lines in the backup.lst file (source file
          of names to backup).

 REVISIONS:
  DATE    NAME    DESCRIPTION
  ----------------------------------------------------------------
 09/11/95 D.KAATZ     INITIAL REVISION
 11/16/95 DK          Put system backup volume label on diskette.

 04/19/96 DK       Display an "in progress" message while accessing
                   the diskette.  Take predefined list of files from
                   text file on the hard drive instead of hard coding it.
 11/02/97 DH       Change procedure to backup to the passed in drive

*****************************************************************
************/
/*
LONG SystemBackup(PSZ pszPath)
*/
LONG  SystemBackup
(
  void
)
{
  /* STATIC VARIABLES  */
   /* NONE */

  /* AUTOMATIC VARIABLES */
  LONG  lRc = RC_OK;
  BOOL  bBreakLoop = FALSE;
  CHAR  szTargetPathFile[ MAXPATH ] = " d:\upgrade\backup:";
  CHAR  szDr[4];           /* drive */
  CHAR  szPath[MAXPATH];    /* path  */
  CHAR  szName[16];         /* name  */
  CHAR  szExt[ 5 ];         /* extension */
  ULONG ulOpCode = 0L; // | DCPY_FAILEAS;  /* Copy op code */
  INT   iTrgtLen = strlen(szTargetPathFile);
  ULONG ulLength;
  APIRET apiRet;
  PEAOP2 EABuf;             /* extended attribute buffer */
```

```
VOLUMELABEL VolLabel;     /* diskette volume label */
SPAFILE spaListFile;

/* FUNCTION BODY */

/* initialize variables */
memset( szDr,   0, sizeof(szDr) );
memset( szPath, 0, sizeof(szPath) );
memset( szName, 0, sizeof(szName) );
memset( szExt,  0, sizeof(szExt) );

//CLEAR_SCREEN;

//GetEnvPath( NG_STAGE_DIR_ENV_VAR, spaListFile.sName );

strcpy( spaListFile.sName, UPGDBKUP_FILE_LIST );
spaListFile.chMode     = READ_ACCESS;
spaListFile.usFileType = FT_TEXT;

lRc = FileOpen( &spaListFile );

if( lRc != RC_OK )
{
  printf( EOL_CHARS );
  printf( STR_LISTFILE_ERR );
  return lRc;
}

EABuf = 0;       /* no extended attributes are defined */

/* disable popup error msgs, if no disk is in A */
DosError( FERR_DISABLEHARDERR );

/* Post message that says job is in progress */

printf(STR_BACKUP_IN_PROGRESS);
fflush(stdout);

/*--------------------------------------*
 * Loop over all system files, attempting
 * to copy them to d:\upgrade\backup.  Ignore any
 * missing system file.
 *--------------------------------------*/

lRc = FileRead( &spaListFile, pszSystemFiles, &ulLength, MAXPATH );

while( bBreakLoop == FALSE && lRc == RC_OK && *pszSystemFiles )
{
  FileSplitPath( pszSystemFiles, szDr, szPath, szName, szExt );
  strcat( szTargetPathFile, szPath  );

  /* Create target directory */
  if( szName[0] == 0 )
  {
```

```
    /* remove trailing '\' character */
    szTargetPathFile[strlen(szTargetPathFile)-1] = 0;

    apiRet = DosCreateDir( szTargetPathFile, EABuf );
    if( apiRet != RC_OK )
    {
      printf(EOL_CHARS);
      printf(STR_DIR_CREATE_ERR, apiRet);
      DosError( FERR_ENABLEHARDERR );
      FileClose( &spaListFile );

      return RC_SYS;
    }

    szTargetPathFile[iTrgtLen] = 0;
  }
  else
  {
    strcat( szTargetPathFile, szName );
    strcat( szTargetPathFile, szExt  );

    apiRet = DosCopy(pszSystemFiles, szTargetPathFile, ulOpCode);
    switch (apiRet)
    {
      case RC_OK:
      case ERROR_FILE_NOT_FOUND:
      case ERROR_PATH_NOT_FOUND:
        /* reset target path */
        szTargetPathFile[iTrgtLen] = 0;
        break;

      default:
      case ERROR_DISK_FULL:
      case ERROR_EAS_NOT_SUPPORTED:
      case ERROR_NEED_EAS_FOUND:
        printf(EOL_CHARS);
        printf(STR_BACKUP_FAIL, apiRet);
        printf(EOL_CHARS);
        printf(CONTACT_INTER_SUPPORT);
        lRc = RC_SYS;
        bBreakLoop = TRUE;
        break;

      case ERROR_DRIVE_LOCKED:
        printf(EOL_CHARS);
        printf( STR_DRIVE_LOCKED );
        lRc = RC_SYS;
        bBreakLoop = TRUE;
        break;

    }
  }

  if( lRc == RC_OK )
```

```
    {
      lRc = FileRead( &spaListFile, pszSystemFiles, &ulLength, MAXPATH );
    }
    if( strncmp(pszSystemFiles, ENDOFFILE, strlen(ENDOFFILE)) == 0 )
    {
      bBreakLoop = TRUE;
    }
  }

  DosError( FERR_ENABLEHARDERR );

  FileClose( &spaListFile );
  printf( EOL_CHARS );
  printf( STR_WARN_ONE );
  printf( EOL_CHARS );
  printf( STR_WARN_TWO );
  fflush(stdout);
  return lRc;
} /* LONG SystemBackup() */
```

1.2.8.4.7.1.20. SystemRestore

```
/*********************************************************************
**********
  FUNCTION:    SystemRestore

  DESCRIPTION:  Restore system files from passed in drive.
                Use the same list, pszSystemFiles, that was used
                to backup the system files.  Just change the
                source drive to the passed in drive:".  Ignore any files that
                don't exist on the floppy.

  ASSUMPTIONS:
                Do not need to create target directories on target drive.
                No blank lines in the backup.lst file (source file
                of names to backup).
                First file named in backup.lst MUST NOT BE config.sys
                or its variations.

  REVISIONS:
  DATE    NAME    DESCRIPTION
  ----------------------------------------------------------------------
 05/15/96 D.KAATZ     INITIAL REVISION - borrowed from ngbackup.c
07/10/96  D.Kaatz
11/02/97  DH  Change procedure to backup to passed in drive instead
                of a:
*********************************************************************
**********/
LONG SystemRestore( PSZ pszPath )
{
  /* STATIC VARIABLES */
```

```
  /* AUTOMATIC VARIABLES */
  LONG    lRc = RC_OK;
  CHAR    szSourcePathFile[MAXPATH];
  ULONG   ulOpCode = 0L | DCPY_FAILEAS|DCPY_EXISTING;  /* Copy op
code */
UCHAR   FSInfoBuf[40];   /* File system info buffer */
  SPAFILE  spaListFile;
  ULONG   ulLength;
  BOOL    bVersion0 = FALSE;// restoring from a version 0 backup diskette?

  /* FUNCTION BODY */

  /* disable popup error msgs, if no disk is in A */
  DosError( FERR_DISABLEHARDERR );

  /***************************
   Open the master list of
   system files file.
   ***************************/
  //GetEnvPath( NG_STAGE_DIR_ENV_VAR, spaListFile.sName );

  strcpy( spaListFile.sName, UPGDBKUP_FILE_LIST );
  spaListFile.chMode    = READ_ACCESS;
  spaListFile.usFileType = FT_TEXT;

  lRc = FileOpen( &spaListFile );
  if( lRc != RC_OK )
  {
    printf( EOL_CHARS );
    printf( STR_LISTFILE_ERR );
    printf( EOL_CHARS );
    printf( CONTACT_INTER_SUPPORT );
    return lRc;
  }

  printf( EOL_CHARS );
  printf(STR_RESTORE_IN_PROGRESS);
  fflush(stdout);

  lRc = FileRead( &spaListFile, pszSystemFiles, &ulLength, MAXPATH );

  while( lRc == RC_OK && *pszSystemFiles )
  {
    /* ---------------------------------------
     * The directory only portions of the list
     * will not be a problem for us, as DosCopy
     * will return ERROR_PATH_NOT_FOUND, which
     * we ignore.
     *---------------------------------------*/
    strcpy( szSourcePathFile, pszSystemFiles );
    szSourcePathFile[0] = 'd:\upgrade\backup';

    lRc = DosCopy( szSourcePathFile, pszSystemFiles, ulOpCode );
    switch (lRc)
```

```
{
  case RC_OK:
  case ERROR_FILE_NOT_FOUND:
  case ERROR_PATH_NOT_FOUND:
    lRc = RC_OK;
    break;

  default:
  case ERROR_DISK_FULL:
  case ERROR_EAS_NOT_SUPPORTED:
  case ERROR_NEED_EAS_FOUND:
    printf( EOL_CHARS );
    printf( STR_RESTORE_FAIL, lRc );
    printf( EOL_CHARS );
    printf( CONTACT_INTER_SUPPORT );
    DosError( FERR_ENABLEHARDERR );
    FileClose( &spaListFile );

    return RC_SYS;

  case ERROR_DRIVE_LOCKED:
    printf( EOL_CHARS );
    printf( STR_DRIVE_LOCKED );
    printf( EOL_CHARS );
    printf( CONTACT_INTER_SUPPORT );
    DosError( FERR_ENABLEHARDERR );
    FileClose( &spaListFile );

    return RC_SYS;
}
lRc = FileRead( &spaListFile, pszSystemFiles, &ulLength, MAXPATH );

//-------------------------------------------------------
// Check if reached end of file.  Getting an EOF return
// code would be ok, but we can't tell normal EOF from
// a premature EOF, so an expected endoffile text in
// the file seems more sure.
//-------------------------------------------------------
if( strncmp(pszSystemFiles, ENDOFFILE, strlen(ENDOFFILE)) == 0 )
{
  *pszSystemFiles = 0;  // set to empty string to break from while loop
}

//-------------------------------------------------------
// If restoring from a version 0 diskette:
//   Don't restore config.sys files.
//-------------------------------------------------------

//---------------------------------------
// Check for config.sys file versions
// Assumption is that they will be listed
// contiguously in the backup.lst file.
//---------------------------------------
if( strstr(strlwr(pszSystemFiles), DEFAULT_CONFIG_SYS_FILE) )
```

```
    {
      // If config file, copy to config.bac.
      strcpy( szSourcePathFile, pszSystemFiles );
      szSourcePathFile[0] = 'd:\upgrade\backup';

      sprintf( pszSystemFiles, "%s%s",
               KNOWN_NG_CONFIG_SYS_PATH,
               BACK_CONFIG_SYS_FILE);
      lRc = DosCopy( szSourcePathFile, pszSystemFiles, ulOpCode );
      lRc = FileRead( &spaListFile, pszSystemFiles,
                      &ulLength, MAXPATH );
    }

  } /* end while lRc == RC_OK && *pszSystemFiles */

  FileClose( &spaListFile );

  DosError( FERR_ENABLEHARDERR );
  if( lRc == RC_OK )
  {
    /*
    ** Rename the sys config report file so it appears again.
    */
    CopyEnvFile (DCMENV_MAIN, SAVED_NG_INI_ASCII_FILE,
          DCMENV_MAIN, DEFAULT_NG_INI_ASCII_FILE);
    DelEnvFile (DCMENV_MAIN, SAVED_NG_INI_ASCII_FILE);

  }
  return lRc;
} /* end of systemrestore */
```

1.2.8.4.7.1.21. CreateUpgradeTmpFile

```
/*****************************************************************
**
  FUNCTION:    CreateUpgradeTmpFile
  AUTHOR:      D.Kaatz
  DESCRIPTION: Read the last successful section from the ng_confg.log
               file, use that to write the tmp file that will
               be used by creat_ng to update the system. Kind of a
               kludgy way to achieve a system update, but this allows
               a large amount of functional leveraging from the
               creat_ng executable.
-----------------------------------------------------------------
 Revision History:
  DATE        COMMENT
-----------------------------------------------------------------
 05/22/96     Copied largely from Wade H's code in field_ng.c.

*****************************************************************
**/
/*
CreateUpgradeTmpFile(PSZ pszPath)
```

```
*/
LONG CreateUpgradeTmpFile
(
  void
)
{
  CHAR  szErrorOut[MAX_ERROR_OUT_LEN + 1];  /* for error output strings
*/
  CHAR  szConfigTag[NG_BUF_IN_LEN + 1];     /* config tag or luggage tag
*/
  CHAR  szOrgConfigTag[NG_BUF_IN_LEN + 1]; /* original config tag or
*/
                                /* luggage tag              */
  CHAR  szCurLine[NG_BUF_IN_LEN + 1];      /* buffer for reading from file */
  PCHAR  pszCurEntry = NULL;              /* ptr used to extract value    */
                                /* current line             */
  USHORT usStrLoc = 0;                  /* used to move in current line */
                                /* buffer                   */
  CHAR  szDownLine1[DOWN_LINE_VAL_LEN + 1]; /* downline #1 value
*/
  CHAR  szDownLine2[DOWN_LINE_VAL_LEN + 1]; /* downline #2 value
*/
  CHAR  szTUpLine[UP_LINE_VAL_LEN + 1];    /* Temp upline value
*/
  CHAR  szTDownLine1[DOWN_LINE_VAL_LEN + 1]; /* Temp downline #1
value     */
  CHAR  szTDownLine2[DOWN_LINE_VAL_LEN + 1]; /* Temp downline #2
value     */
  CHAR  szUpLine[UP_LINE_VAL_LEN + 1];    /* upline value             */

  PCHAR pszTemp = NULL;                 /* used for string fxns        */
  CHAR  szNGCfgLogFullPath[MAXPATH + 1];  /* complete path for the
*/
                                /* nextgen config log file      */
  BOOL  fNGCfgLogFileFound = FALSE;       /* set if NG cfg log file exists */
  CHAR  szNGTempCfgPath[MAXPATH + 1];     /* complete path for the
*/
                                /* nextgen temp config file     */
  BOOL  fHiddenFile = FALSE;            /* set if hidden file          */
  LONG  lFileRC = RC_OK;                /* used in call to FileExist    */
  FILE* pfCfgLog = NULL;                /* file ptr for NG config log   */
  FILE* pfTempCfg = NULL;               /* file ptr for temp NG config  */
  BOOL  fExit = FALSE;                  /* set to true if error requires */
                                /* program termination          */
  BOOL   fSerialCrd = FALSE;            /* FLAGS                      */
  BOOL   fAddEther  = FALSE;            /* FLAGS                      */
  BOOL   fUpFound     = FALSE;          /* FLAGS                      */
  BOOL   fDown1Found  = FALSE;          /* FLAGS                      */
  BOOL   fDown2Found  = FALSE;          /* FLAGS                      */
  BOOL   fPort1Found  = FALSE;          /* FLAGS                      */
  BOOL   fPort2Found  = FALSE;          /* FLAGS                      */
  BOOL   fSerialFound = FALSE;          /* FLAGS                      */
  BOOL   fOrgCfgTagFound = FALSE;       /* FLAGS                      */
                                /* matches DCS 300 version     */
```

```
memset(szConfigTag,       NUL, NG_BUF_IN_LEN + 1);
memset(szOrgConfigTag,    NUL, NG_BUF_IN_LEN + 1);
memset(szNGCfgLogFullPath, NUL, MAXPATH + 1);
memset(szNGTempCfgPath,   NUL, MAXPATH + 1);
memset(szDownLine1,  NUL, DOWN_LINE_VAL_LEN + 1);
memset(szDownLine2,  NUL, DOWN_LINE_VAL_LEN + 1);
memset(szUpLine,     NUL, UP_LINE_VAL_LEN  + 1);
memset(szTDownLine1, NUL, DOWN_LINE_VAL_LEN + 1);
memset(szTDownLine2, NUL, DOWN_LINE_VAL_LEN + 1);
memset(szTUpLine,    NUL, UP_LINE_VAL_LEN  + 1);

/****************************************
* Open of the NextGen                   *
* DCS 300 config log file.              *
****************************************/
pszTemp = szNGCfgLogFullPath;
sprintf(szNGCfgLogFullPath,"%s",KNOWN_NG_SYS_INI_PATH);
if (szNGCfgLogFullPath[strlen(pszTemp)-1] != OSCHAR_DIR)
  szNGCfgLogFullPath[strlen(pszTemp)] = OSCHAR_DIR;
strcat(szNGCfgLogFullPath,DEFAULT_NG_CONFIG_LOG_FILE);
pszTemp = NULL;

lFileRC = FileExists(szNGCfgLogFullPath, &fNGCfgLogFileFound,
                                          &fHiddenFile);

if ( (lFileRC == RC_OK) && (fNGCfgLogFileFound == TRUE) )
{
  /*************************************
  * File exists; open it for read mode  *
  *************************************/
  pfCfgLog = fopen(szNGCfgLogFullPath,FO_READ_ONLY);

}
else
{
  /*************************************
  * File does not exist; this is an     *
  * error; Must contact Intermec        *
  * support                             *
  *************************************/
  sprintf(szErrorOut,"%s\n%s\n%s\n",ERROR_HEADING_300_D,
                    NO_CONFIG_LOG_FILE_FOUND,
                    CONTACT_INTER_SUPPORT);
  printf("\n\n%s\n",szErrorOut);
  fflush(stdout);

  /***************
  * Set to exit  *
  ***************/
  return RC_EXIT;
}
```

```
if (pfCfgLog != NULL)
{
  /************************************
  * Now read in NG config log file to  *
  * determine what is currently in the *
  * system with respect to upline and  *
  * downline cards as well as serial   *
  * and RF cards.                      *
  * This info should only be written   *
  * to the NG config log file if a     *
  * configuration has been completed   *
  * successfully, however care should  *
  * be taken to get the results from   *
  * the LAST SUCCESSFUL configuration. *
  * Get the last entries               *
  * since this file is appended to as  *
  * a DCS 300's config is changed.  *
  ************************************/
  memset(szCurLine,NUL,NG_BUF_IN_LEN + 1);

  while (fgets(szCurLine,NG_BUF_IN_LEN,pfCfgLog) != NULL)
  {
    /*********************************
    * Strip out all white space from  *
    * current line                    *
    *********************************/
    StripCRNL(szCurLine);
    usStrLoc = (USHORT) strspn(szCurLine,szWhites);
    pszCurEntry = szCurLine;
    pszCurEntry = pszCurEntry + (int) usStrLoc;

    /*********************************
    * Remove any successive spaces    *
    *********************************/
    RemSuccessiveDelims(pszCurEntry,SP);

    if ( strncmp(StrToUpr(pszCurEntry),
                 StrToUpr(NG_CONFIG_TAG_EQU_STR),
                 strlen(NG_CONFIG_TAG_EQU_STR)) == 0 )
    {
      // *******************************
      // Check that the following
      // configuration data is for
      // the same version as this
      // field upgrade
      // Do this by looking at the
      // configuration (luggage) tag.
      // For upgrades, we don't expect the
      // configuration data to be the same
      // version as we want to upgrade to.
      // *******************************
      pszCurEntry = pszCurEntry +
                          strlen(NG_CONFIG_TAG_EQU_STR);
      usStrLoc = (USHORT) strspn(pszCurEntry,szWhites);
```

```
  pszCurEntry = pszCurEntry + (int) usStrLoc;

  // ******************
  // Valid luggage tag? *
  // ******************
  if ( strncmp(pszCurEntry, MODEL_300_LUG_PREFIX,
              strlen(MODEL_300_LUG_PREFIX)) == 0)
  {
    // ************************************
    // * Now get the original config tag    *
    // ************************************
    strncpy( szOrgConfigTag,
          pszCurEntry,
          strlen(NG_CONFIG_TAG_EQU_STR));
    fOrgCfgTagFound = TRUE;
  }
}

if ( strncmp(StrToUpr(pszCurEntry),NG_CONFIG_UPLINE_STR,
                  strlen(NG_CONFIG_UPLINE_STR)) == 0 )
{
  pszCurEntry = pszCurEntry + strlen(NG_CONFIG_UPLINE_STR);
  usStrLoc = (USHORT) strspn(pszCurEntry,szSkips);
  pszCurEntry = pszCurEntry + (int) usStrLoc;

  strncpy(szTUpLine, pszCurEntry, UP_LINE_VAL_LEN);

  if ( (strcmp(szTUpLine,ETHER_UP) == 0) ||
    (strcmp(szTUpLine,TOKEN_UP) == 0) ||
    (strcmp(szTUpLine,TWINAX_UP) == 0) ||
    (strcmp(szTUpLine,COAX_UP) == 0) ||
    (strcmp(szTUpLine,SDLC_UP) == 0) )
  {
    fUpFound   = TRUE;
  }
  else
  {
    ;
  }


}
else if ( strncmp(StrToUpr(pszCurEntry),NG_CONFIG_DOWN1_STR,
                  strlen(NG_CONFIG_DOWN1_STR)) == 0 )
{
  pszCurEntry = pszCurEntry + strlen(NG_CONFIG_DOWN1_STR);

  usStrLoc = (USHORT) strspn(pszCurEntry,szSkips);
  pszCurEntry = pszCurEntry + (int) usStrLoc;

  strncpy(szTDownLine1,pszCurEntry,DOWN_LINE_VAL_LEN);
  fDown1Found   = TRUE;
```

```
}
else if ( strncmp(StrToUpr(pszCurEntry),NG_CONFIG_DOWN2_STR,
                  strlen(NG_CONFIG_DOWN2_STR)) == 0 )
{
  pszCurEntry = pszCurEntry + strlen(NG_CONFIG_DOWN2_STR);

  usStrLoc = (USHORT) strspn(pszCurEntry,szSkips);
  pszCurEntry = pszCurEntry + (int) usStrLoc;

  strncpy(szTDownLine2,pszCurEntry,DOWN_LINE_VAL_LEN);
  fDown2Found   = TRUE;


}
else if ( strncmp(StrToUpr(pszCurEntry),
          NG_CONFIG_DOWN1_PORT_STR,
          strlen(NG_CONFIG_DOWN1_PORT_STR)) == 0 )
{
  pszCurEntry = pszCurEntry +
                      strlen(NG_CONFIG_DOWN1_PORT_STR);

  usStrLoc = (USHORT) strspn(pszCurEntry,szSkips);
  pszCurEntry = pszCurEntry + (int) usStrLoc;

  if ( strcmp(pszCurEntry,TRUE_STR) == 0)
  {
    fPort1Found   = TRUE;
  }
  else if ( strcmp(pszCurEntry,FALSE_STR) == 0)
  {
    fPort1Found   = TRUE;
  }
  else
  {
    /*****************************
    * Error                      *
    *****************************/
    ;
  }

}
else if ( strncmp(StrToUpr(pszCurEntry),
           NG_CONFIG_DOWN2_PORT_STR,
          strlen(NG_CONFIG_DOWN2_PORT_STR)) == 0 )
{
  pszCurEntry = pszCurEntry +
            strlen(NG_CONFIG_DOWN2_PORT_STR);

  usStrLoc = (USHORT) strspn(pszCurEntry,szSkips);
  pszCurEntry = pszCurEntry + (int) usStrLoc;

  if ( strcmp(pszCurEntry,TRUE_STR) == 0)
  {
    fPort2Found   = TRUE;
```

```
  }
  else if ( strcmp(pszCurEntry,FALSE_STR) == 0)
  {
    fPort2Found   = TRUE;
  }
  else
  {
    /*****************************
    * Error                 *
    *****************************/
    ;
  }

}
else if ( strncmp(StrToUpr(pszCurEntry),NG_CONFIG_SERIAL_STR,
                  strlen(NG_CONFIG_SERIAL_STR)) == 0 )
{
  pszCurEntry = pszCurEntry + strlen(NG_CONFIG_SERIAL_STR);

  usStrLoc = (USHORT) strspn(pszCurEntry,szSkips);
  pszCurEntry = pszCurEntry + (int) usStrLoc;

  if ( strcmp(pszCurEntry,TRUE_STR) == 0)
  {
    /*****************************
    * Serial card installed.    *
    *****************************/
    fSerialCrd = TRUE;
    fSerialFound = TRUE;
  }
  else if ( strcmp(pszCurEntry,FALSE_STR) == 0)
  {
    fSerialCrd = FALSE;
    fSerialFound = TRUE;
  }
  else
  {
    /*****************************
    * Error                 *
    *****************************/
    ;
  }
}

/*********************************
* This next else checks for a success string in the config   *
* log file.  A known potential problem could be if one of the *
* above six values was missing in the currently read config   *
* but had been previously read  in a chronologically earlier   *
* entry in the ng_confg.log file.
* The old value would be used as the value for this config     *
* parameter. This is especially  true for the BOOL values      *
* (fSerialCrd,fPort1Found, fPort2Found).                 *
```

```
*********************************/

else if ( (fUpFound == TRUE) && (fDown1Found == TRUE) &&
          (fDown2Found == TRUE) &&
          (fPort1Found == TRUE) && (fPort2Found == TRUE) &&
          (fSerialFound == TRUE) &&
          (fOrgCfgTagFound == TRUE) )
{
  /*************************************
  * Have found values for every parameter *
  * Now make sure that next line        *
  * indicates that this was a successful *
  * config, if not throw the stuff away  *
  * and keep going                      *
  *************************************/

  if (strncmp(StrToUpr(pszCurEntry),
         StrToUpr(NG_CONFIG_SUCCESS_STR),
          strlen(NG_CONFIG_SUCCESS_STR)) == 0)
  {
    /***********************************
    * Found a success string. Store     *
    * values for upline and downlines   *
    * and set flag for good read        *
    ***********************************/
    strncpy(szUpLine,szTUpLine,UP_LINE_VAL_LEN);
    strncpy(szDownLine1,szTDownLine1,DOWN_LINE_VAL_LEN);
    strncpy(szDownLine2,szTDownLine2,DOWN_LINE_VAL_LEN);

  }
  else
  {
    ;
  }

  /************************
  * reset for next pass  *
  ************************/
  fUpFound      = FALSE;
  fDown1Found   = FALSE;
  fDown2Found   = FALSE;
  fPort1Found   = FALSE;
  fPort2Found   = FALSE;
  fSerialFound  = FALSE;
  fOrgCfgTagFound = FALSE;

 }
 } // end of while (fgets(szCurLine,NG_BUF_IN_LEN,pfCfgLog) != NULL)
} // end of if pfCfgLog != NULL

/*************************************
* Open the temporary field update file *
*************************************/
if (fExit != TRUE)
```

```
{
  pszTemp = szNGTempCfgPath;
  sprintf(szNGTempCfgPath, "%s%c%s", E_DRIVE,
                                OSCHAR_DIR, UPGRADE_DIR);
  if (szNGTempCfgPath[strlen(pszTemp)-1] != OSCHAR_DIR)
    szNGTempCfgPath[strlen(pszTemp)] = OSCHAR_DIR;
  strcat(szNGTempCfgPath, DEFAULT_NG_TEMP_UPGRAD_FILE);
  pszTemp = NULL;


  pfTempCfg = fopen(szNGTempCfgPath, FO_WRITE);

  if (pfTempCfg == NULL)
  {
    /*********************************
    * Error can't open temp cfg file  *
    * for writing of field update     *
    * info                            *
    *********************************/
    sprintf(szErrorOut,"%s\n%s\n%s\n",ERROR_HEADING_300_D,
                      NG_TEMP_UPGRAD_FILE_ERR,
                      CONTACT_INTER_SUPPORT);
    printf("\n\n%s\n",szErrorOut);
    fflush(stdout);

    fExit = TRUE;
    fclose(pfCfgLog);
    return RC_EXIT;
  }
}
if (fExit != TRUE)
{
  /*************************************
  * Now build the the config tag string *
  * Note use of "%3.3s" for the output  *
  * of the MODEL_300_LUG_PREFIX         *
  *************************************/
  if (fExit != TRUE)
  {
    sprintf(szConfigTag,
    "%s%s%s",
    MODEL_300_LUG_PREFIX,VERSION_STR,
    &szOrgConfigTag[strlen(MODEL_300_LUG_PREFIX) +
                                          strlen(VERSION_STR)]);
  }

  /*********************************
  * Put info into temp config file  *
  * this will be read in by the     *
  * creat_ng.exe and used to set    *
  * up the DCS 300 for the new      *
  * field updates                   *
  *********************************/
  fprintf(pfTempCfg,"%s\n",szConfigTag);
```

```
    fprintf(pfTempCfg,"%d\n",(int) fSerialCrd);
    fprintf(pfTempCfg,"%d\n",(int) fAddEther);
  }
  fclose(pfCfgLog);
  fclose(pfTempCfg);
  return RC_OK;

} /* end of CreateUpgradeTmpFile() */
```

1.2.8.4.7.1.22. DosExecPgmWrapper

```
/*********************************************************************
*********
  FUNCTION:    DosExecPgmWrapper
  AUTHOR:      D. Kaatz
  DESCRIPTION:  A generic way to call the DosExecPgm function.
                Pass in the program to call and a string of command line
                inputs.
  RETURNS:     Result of call to DosExecPgm

 REVISIONS:
  DATE    NAME    DESCRIPTION
  ----------------------------------------------------------------
 06/25/96 D.KAATZ    INITIAL REVISION
*********************************************************************
*********/
LONG DosExecPgmWrapper(PCHAR szPgm, PCHAR szArgs)
{
  CHAR      szObjectBuffer[FILES_GP_NAME_SIZE];
  RESULTCODES Results;
  PSZ       pszArgList;     /* Formatted argument list for DosExecPgm */
  PSZ       pszTemp;        /* Temporary argument buffer */
  LONG      lRc;

  pszArgList = (PSZ)MemAlloc(FILES_GP_NAME_SIZE + 3);

  if (pszArgList == NULL)
  {
    lRc = RC_MEM; /* system out of memory */
    printf( "\nSystem out of memory." );
    return lRc;
  } /* end if <pszArglist> = null */

  /*-------------------------------------------------------*/
  /* Initialize the argument list for the DosExecPgm       */
  /* => the program name is first, followed by the arguments  */
  /* space separated, and double NUL terminated.           */
  /* <pgm name> NUL <arg-list> NUL NUL                     */
  /*-------------------------------------------------------*/
  strcpy (pszArgList, "");
  // Store the process name in the argument list
  pszTemp = StrInsStr(pszArgList, 0, szPgm);
```

```
    // Append a null after the process name
    pszTemp = StrInsChr(pszArgList, (USHORT)strlen(szPgm), NUL, 1);
    // Store the passed argument string in the location
    // after the process name and NUL
    StrInsStr(pszTemp, 0, szArgs);
    // Set the double NUL's at the end of the argument list
    StrInsChr(pszTemp, (USHORT)strlen(szPgm), NUL, 2);
    lRc = (LONG)DosExecPgm(
            szObjectBuffer,
            FILES_GP_NAME_SIZE,
            EXEC_SYNC,        // synchronous execution
            pszArgList,       // command line input to the exe
            0,
            &Results,         // results from running pgm.
            szPgm);

    MemFree( pszArgList );
    return lRc;
  } /* end of DosExecPgmWrapper */
```

1.2.8.4.7.2. Pseudo-code for upgrade.h

```
/*****************************************************************************
*
* FILE NAME:  upgrade.h
*
* PURPOSE:    This is the include file for upgrade.c
*             This file also includes the prototypes and variables
*             needed by other subsystems (e.g., protoypes for VSE subsystem).
*
* AUTHOR:     D. Hughes
*
* DATE:       08/15/97
*
* COPYRIGHT (c) 1997 INTERMEC CORPORATION, ALL RIGHTS RESERVED
*
*****************************************************************************/
/*
These will now be passed in paths.
#define UPGRADE_PATH     "d:\\upgrade"
#define UNZIP_EXE      "e:\\tools\\UNZIP.EXE"
#define D_IMAGE        "-qq d:\\upgrade\\d_image.zip -d D:\\"
#define C_IMAGE        "-qq d:\upgrade\\\c_image.zip -d C:\\"
*/

#define LOG_FILE      "d:\\upgrade\\upgrad.log"
#define PHASE         "PHASE="
#define TEMP_NAME_PATH "a:\\tempfile.bac"
//#define MAXPATH       80
#define NG_BUF_IN_LEN. 280

#define C_DRIVE             "C: "
#define D_DRIVE             "D: "
```

```
#define CD_ROM_DRIVE  "G:"

#define COPY_COMMAND     "COPY "
#define FDISK_COMMAND  "FDISK "

#define REDIRECT_OUTPUT ">"
#define REDIRECT_INPUT  "<"
#define INPUT_FILE     "inputfil.txt"
#define NUL_FILE       "nul"

#define MAX_ERROR_OUT_LEN  560

#define RC_FAIL_OPEN  -1
#define ERROR_HEADING_300_D          "Model DCS 300 - "
#define CONTACT_SUPER_ABORT          "Contact supervisor - Aborting."
#define CONTACT_SUPER                "Contact supervisor."
#define CONTACT_INTER_SUPPORT        "Contact Intermec Support."
#define COPY_C_DRIVE_ERR             "Error Copying files to C: Drive"
#define COPY_D_DRIVE_ERR             "Error Copying files to D: Drive"

#define  STR_BACKUP_FAIL             "Backup failure: #%ld"
#define  STR_RESTORE_FAIL            "Restore failure: #%ld"

#define  STR_LISTFILE_ERR            "Could not open master system file list."
#define  STR_RESTORE_IN_PROGRESS     "Restore is in progress..."
#define  STR_BACKUP_IN_PROGRESS      "Backup is in progress..."
#define  STR_DIR_CREATE_ERR          "Target directory creation error.  #%ld"
#define  STR_PRESS_KEY               "Press any key when ready. "
#define  STR_ACCESS_DENIED           "ERROR - Access to drive denied"
#define  STR_MIGRATELIST_ERR         "Could not open the migration list file."

#define CLEAR_SCREEN   printf("\x1b[2J")

#define SECTOR       512

//#define FO_READ_ONLY  "rb"
//#define FO_WRITE      "wb"

#define  SYSTEM_BACKUP_LABEL         "SYSBACKUP1"
#define  SYSTEM_BACKUP_LABEL_LEN 10        // length of backup label

#define  SYSTEM_BACKUP_LABEL_0       "SYS_BACKUP"  // Version 0 backup label

/*
These will be passed in paths
#define  UPGDBKUP_FILE_LIST          "d:\\upgrade\\upgdbkup.lst"  // name of file containing
                                     // files to backup
#define  SYSTEM_FILE_LIST            "d:\\upgrade\\restore.lst" // name of file containing
                                     // list of files to restore.
#define  MIGRATE_FILE_LIST           "d:\\upgrade\\migrate.lst" // name of file containing list
                                     // of files to migrate from
                                     // a previous DCS 300 version.
*/
```

```
#define  ENDOFFILE              "ENDOFFILE"  // should be last string in .lst file
#define  MIGRATE_ZERO             "MIGRATE_0"  // migration info indicator in file
#define  CREAT_NG_EXE             "D:\\NEXTGEN\\CREAT_NG.EXE"

#define  BACKUP      1
#define  RESTORE     2

#define MINOR       0
#define REBOOT      1
#define SHUTDOWN 2
#define SEVERE      3

#define ALT_BOOT_DRIVE 'f'

/*
** End of file upgrade.h
** Copyright (c) 1997  Intermec Corp.  All rights reserved.
*/
```

1.2.9. Data Organization

Changes will be made to the SysMaintitems structure in ngextrn.h

```
SBMENUITEM SysMaintItems[] =
     {{"Configure Download Server"     , DB_SET_TERMDOWNLOAD,
                                         DB_SET_TERMDOWNLOAD_eh},
      {"Reset to Factory Defaults"     , DB_RESET_DEFAULTS,
                                         DB_RESET_DEFAULTS_eh},
      {"Back up System Files"          , DB_SYSBAK, DB_SYSBAK_eh},
      {"Restore System Files"          , DB_SYSRSTR, DB_SYSRSTR_eh},
      {"Terminal License Upgrade"      , DB_LICENSE,DB_LICENSE_eh},
      {"Screen Mapping License Upgrade" , DB_LICENSE, DB_LICENSE_eh},
      {"Send Transactions"             , DB_SENDTRAN, DB_SENDTRAN_eh},
      {"Receive Transactions"          , DB_APPLICATION, 0},
      {"Electronic Software Distribution"      , DB_ELECT_SOFT_DISTRIB, 0},
      {"Install Accessories"           , DB_INSTALL_ACC, DB_INSTALL_ACC_eh},
      {"Start Host Session"            , DB_SET_SESSION, DB_SET_SESSION_eh},
      {"Terminal Password Configuration", DLG_222, DLG_222_eh},
      {"Controller Command Prompt"     , DB_SET_COMMAND_PASSWORD,
                                         DB_SET_COMMAND_PASSWORD_eh},
      {"DCS Upgrade Utility", DB_300_UPG_UTIL, DB_300_UPG_UTIL_ef},
      {"", 0, 0 }};
```